THE UNIVERSITY
of EDINBURGH

Dynamic Reconfiguration Frameworks for High-Performance Reliable Real-Time Reconfigurable Computing

Adewale Adetomi

A thesis submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

The University of Edinburgh

October 2018

"Many shall run to and fro,
and knowledge shall be
increased"

Daniel 12:4

This thesis ...

... a notch in the circle of knowledge

Declaration of Originality

I hereby declare that this thesis was composed and originated entirely by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualifications.

Adewale Adetomi
June 2019
Edinburgh, UK

Acknowledgements

First and foremost, all glory, honour, and adoration to God, my heavenly Father, my Saviour, Jesus Christ, and my Helper, the Holy Spirit, who is the embodiment of all knowledge. May His name be forever praised for the inspirations for this work, for courage when there seemed to be no hope, for strength when the journey was tough, and for the grace to finish well.

Loving appreciation goes to my sweetheart and friend, Titilayo Adetomi, for her unperturbed support and understanding. May the Lord give you a swell reward both here on earth and in eternity. Of course, much love to my pleasant kids, Daniel and Peniel Adetomi. May God keep you and may His Spirit rest upon you.

I would also like to thank my supervisor, Prof. Tughrul Arslan, for his immense guidance and visionary backing. Your astute leadership and man management skills are inestimable. Thanks a million for the mentoring and opportunities. Appreciation to my friend, Godwin Enemali for his constant support. Indeed, you are a friend like a brother. A million thanks to all the members of the EWireless Research Group at the University of Edinburgh for creating a friendly research environment.

I will not forget to mention my family back home in Nigeria. Thank you Daddy and Mummy Adetomi. I remember your sacrifices in bringing me and my siblings up. God is aware of this and He will surely reward you. Please, keep well and stay blessed. My gratitude also goes to my in-laws. Thank you for your prayers and support.

To my church members at the RCCG King of Glory Edinburgh, thank you for the fellowship. To my friends, Gbenga Adekunle and other, thank you for being there. I am indebted to the National Universities Commission, Nigeria and the Petroleum Development Trust Fund, Nigeria for funding this work under the Presidential Scholarship Scheme for Innovation and Development. To all others not mentioned but who have in one way or the other contributed to the successful completion of this piece of work, accept my sincere gratitude.

Lay Summary

Advancement in technology over the years has led to improvement in the performance of modern reconfigurable computing devices, making them more attractive in high-end application domains like aerospace, defence, military, and nuclear power stations; thanks to technology giants like Microsoft, Amazon, and Baidu, which are now embracing these devices to meet their critical computing needs. Because of the critical nature of these application domains, these devices and the computations on them are required to be highly reliable. As such, algorithms and frameworks developed for application on these devices must be implemented with sufficient considerations for reliability.

Meanwhile, the capabilities of these modern reconfigurable devices as computing platforms can be better exploited by providing supporting frameworks to manage the on-chip resources. Existing approaches in this regard have provided key headways but they are limited in reliability, security, and efficiency, especially regarding the flexible use of the on-chip resources for computation.

This work proposes the enabling frameworks for more *reliable*, *secure*, and *efficient* computing on modern reconfigurable devices. For evaluation, a case study that uses a NASA Jet Propulsion Laboratory's spectrometer data processing application is employed to demonstrate the improved reliability possible. It is observed that up to 74% time saving can be achieved for reliability by error mitigation when compared to state-of-the-art vendor implementations. Moreover, an improvement in overall system reliability is observed when the proposed frameworks are deployed in the data processing application. In addition, for the secure configuration of a device, a time saving of up to 32% or 83% is achieved, depending on the device family; and on-chip resource usage savings in excess of 90% compared to state-of-the-art.

Abstract

The sheer hardware-based computational performance and programming flexibility offered by reconfigurable hardware like *Field-Programmable Gate Arrays* (FPGAs) make them attractive for computing in applications that require *high performance*, *availability*, *reliability*, *real-time processing*, and *high efficiency*. Fueled by fabrication process scaling, modern reconfigurable devices come with ever greater quantities of on-chip resources, allowing a more complex variety of applications to be developed. Thus, the trend is that technology giants like Microsoft, Amazon, and Baidu now embrace reconfigurable computing devices likes FPGAs to meet their critical computing needs. In addition, the capability to autonomously reprogramme these devices in the field is being exploited for reliability in application domains like aerospace, defence, military, and nuclear power stations. In such applications, real-time computing is important and is often a necessity for reliability. As such, applications and algorithms resident on these devices must be implemented with sufficient considerations for real-time processing and reliability.

Often, to manage a reconfigurable hardware device as a computing platform for a multiplicity of homogenous and heterogeneous tasks, reconfigurable operating systems (ROSes) have been proposed to give a *software look* to hardware-based computation. The key requirements of a ROS include partitioning, task scheduling and allocation, task configuration or loading, and inter-task communication and synchronization. Existing ROSes have met these requirements to varied extents. However, they are limited in reliability, especially regarding the flexibility of placing the hardware circuits of tasks on device's chip area, the problem arising more from the partitioning approaches used. Indeed, this problem is deeply rooted in the static nature of the on-chip inter-communication among tasks, hampering the flexibility of runtime task relocation for reliability.

This thesis proposes the enabling frameworks for *reliable*, *available*, *real-time*, *efficient*, *secure*, and *high-performance* reconfigurable computing by providing techniques and mechanisms for reliable runtime *reconfiguration*, and dynamic inter-

circuit *communication* and synchronization for circuits on reconfigurable hardware. This work provides task configuration infrastructures for reliable reconfigurable computing. Key features, especially reliability-enabling functionalities, which have been given little or no attention in state-of-the-art are implemented. These features include internal register read and write for device diagnosis; configuration operation abort mechanism, and tightly integrated selective-area scanning, which aims to optimize access to the device's reconfiguration port for both task loading and error mitigation.

In addition, this thesis proposes a novel reliability-aware inter-task communication framework that exploits the availability of dedicated clocking infrastructures in a typical FPGA to provide inter-task communication and synchronization. The clock buffers and networks of an FPGA use dedicated routing resources, which are distinct from the general routing resources. As such, deploying these dedicated resources for communication sidesteps the restriction of static routes and allows a better relocation of circuits for reliability purposes.

For evaluation, a case study that uses a NASA/JPL spectrometer data processing application is employed to demonstrate the improved reliability brought about by the implemented configuration controller and the reliability-aware dynamic communication infrastructure. It is observed that up to 74% time saving can be achieved for selective-area error mitigation when compared to state-of-the-art vendor implementations. Moreover, an improvement in overall system reliability is observed when the proposed dynamic communication scheme is deployed in the data processing application.

Finally, one area of reconfigurable computing that has received insufficient attention is security. Meanwhile, considering the nature of applications which now turn to reconfigurable computing for accelerating compute-intensive processes, a high premium is now placed on security, not only of the device but also of the applications, from loading to runtime execution. To address security concerns, a novel secure and efficient task configuration technique for task relocation is also investigated, providing configuration time savings of up to 32% or 83%, depending on the device; and resource usage savings in excess of 90% compared to state-of-the-art.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

AES	Advanced Encryption Standard
ASIC	Application-Specific Integrated Circuit
ATAL	Advance Task Address loading
BBRAM	Battery-Backed Random Access Memory
BPI	Byte Peripheral Interface
BRAM	Block Random Access Memories
CAM	Configuration Memory (CMEM) Access Manager
CB	Connection Box
CBC	Cypher Block Chaining
CDC	Clock Domain Crossing
CE	Clock Enable
CELOC	Clock-Enabled Low-Overhead Communication
CERANoC	Clock-Enabled Relocation-Aware Network on Chip
CFG	Configuration
CFRT	Configuration Frame Readback Template
CFWT	Configuration Frame Write Template
CIRIS	Compositional InfraRed Imaging Spectrometer
CLB	Configurable Logic Block
CMA	Cumulative Moving Average
CMC	Configuration Monitoring Circuit
CMEM	Configuration Memory
CMOS	Complementary Metal-Oxide-Semiconductor
COBS	Consistent-Overhead Byte Stuffing
CONS	Consistent-Overhead Nibble Stuffing
COTS	Commercial off The Shelf
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CR	Circuit Region

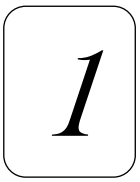
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DPR	Dynamic Partial Reconfiguration
DRP	Dynamic Reconfiguration Port
DSP	Digital Signal Processor
DWC	Decrypt Word Count
ECC	Error Correction Code
EDF	Earliest Deadline First
FAR	Frame Address Register
FDC	FAR Detection Circuit
FDRI	Frame Data Register Input
FF	Flip-Flop
FMC	FAR Modification Circuit
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
FSS	Frame Synchronization Sequence
FTS	Fourier Transform Spectrometer
GPU	Graphic Processing Unit
HCE	Hot-Carrier Effect
HMAC	Hash Message Authentication Code
HWμK	Hardware Microkernel
IBUF	ICAP Buffer
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
IDE	Integrated Development Environment
IFSM	ICAP Finite State Machine
IOB	Input/Output Block
IP	Intellectual Property
IV	Initial Vector
JTAG	Joint-Test Action Group
LUT	Look-Up Table
MAC	Message Authentication Code

MFW	Multiple Frame Write
NBTI	Negative Bias Thermal Instability
NoC	Network on Chip
OES	Operation Ending Sequence
OS	Operating System
OSS	Operation Starting Sequence
P2P	Point-to-Point
PAL	Programmable Array Logic
PB	Partial Bitstream
PBR	Partial Bitstream Relocation
PIP	Programmable Interconnect Point
PISO	Parallel-In Serial-Out
PL	Programmable Logic
PLL	Phase-Locked Loop
PR	Partial Reconfiguration
PS	Processing System
PUF	Physically Unclonable Function
RAM	Random Access Memory
RC	Reconfigurable Computing
RHBD	Radiation Hardening By Design
RHBP	Radiation Hardening By Process
RM	Reconfigurable Module
ROS	Reconfigurable Operating System
RP	Reconfigurable Partition
RSA	Rivest-Shamir-Adleman
SART	Selective Alternate Routing Technique
SB	Switch Box
SEB	Single Event Burnout
SEC-DED	Single-Error Correction, Double-Error Detection
SEE	Single-Event Effect
SEFI	Single-Event Functional Interrupt
SEGR	Single Event Gate Rupture

SEL	Single Event Latch-up
SEM	Soft Error Mitigation
SERDES	Serializer-Deserializer
SET	Single-Event Transient
SEU	Single-Event Upset
SHA	Secure Hash Algorithm
SIPO	Serial-In Parallel-Out
SNR	Signal-to-Noise Ratio
SoC	System on Chip
SoC	System-on-Chip
SPA	Simple Power Analysis
SPI	Serial Peripheral Interface
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SRL	Shift Register LUT
SWμK	Software Microkernel
TDDB	Time-Dependent Dielectric Breakdown
TID	Total Ionization Dose
TIL	Task Interface Logic
TMR	Triple Modular Redundancy
UART	Universal Asynchronous Receiver-Transmitter
WCET	Worst-Case Execution Time
WL	Wear Levelling
XADC	Xilinx Analogue-to-Digital Converter

Chapter 1

Introduction



There is a surge in the uptake of reconfigurable hardware like *Field-Programmable Gate Arrays* (FPGAs) as computing devices, with this idea being referred to as *Reconfigurable Computing* (RC). Application domains ranging from datacentres [1] to aerospace now turn to RC to meet their high-performance computing needs. The increase in the density and on-chip resources [2] of modern FPGAs has made them even a more capable platform for computation. FPGAs are now deployed to execute important tasks in space applications (for example, satellites), aircraft, and datacentres [3][4]. Considering the high-end nature of these applications, there is a need for reliability, security, and ultimately, abstraction tools to ease the development of FPGA-based applications. In fact, the most important limiting factors in the adoption of FPGAs as computing platforms are the *difficulty in designing applications offline*, and *managing their execution in run time*. The design of applications requires expert knowledge in *Hardware Description Languages* (HDLs). Moreover, HDL code development, testing, and debugging on available synthesis and simulation tools are far from being trivial.

Over the past few decades, computing has been heavily software-based, with high-level software routines written in C and other languages executing on traditional processors. However, the sequential fetch-decode-execute nature of the von Neumann architecture, coupled with memory access bottleneck, limits the performance of processors [5]. Fuelled by transistor scaling (*Moore's Law*) [6][7], over the years, processor speed has increased significantly, but memory chips have not been able to keep up, with processors forced to spend most of their time idling on *wait cycles* for memory access. Several techniques like caching, branch prediction, and pipelining [8] have been advanced to alleviate this bottleneck. Even with multiple processor cores fabricated on a single chip to parallelize computation and thus improve throughput, the inherent sequential nature of CPUs means that only a limited increase in throughput can be obtained.

Despite the obvious limitations of processors and the advantages of hardware, a wholesale change in computing paradigm from *software-based* to *hardware-based* seems far-fetched, considering that mainstream computing facilities currently depend heavily on conventional processors. Moreover, processors excel in certain classes of application that hardware computation struggles with, for example, control algorithms; and are superior at context switching, able to switch program execution in a matter of a few clock cycles [9]. On the other hand, what is observed is that the use of reconfigurable hardware to accelerate compute-intensive and mission-critical tasks is gaining traction. This presents a unique opportunity to explore the immense capability offered by reconfigurable hardware, especially from the point of view of high-level application development.

Over the years, specialized hardware like *Application-Specific Integrated Circuits* (ASICs) have been used to speed up computation, with the *Graphic Processing Unit* (GPU) being a good example. While ASICs offer advantages such as very low power consumption and cost effectiveness when mass produced, the lack of flexibility means they have limited application. On the other hand, though FPGAs are more suitable for low to medium scale production, the provision of programmability means they can be adapted for use in a wide range of applications. Moreover, modern FPGAs have rivalled ASICs in terms of power consumption [10].

1.1 Motivation and Justification

The closest programming languages to the bare-metal processor are machine codes and assembly language but programming a processor in any of these is *tedious*, *time-consuming*, and *error-prone* [11]. This has compelled the development of high-level languages, which require compilers to convert the high-level code to machine-readable instructions for execution on the processor. As a result, the description of the hardware underlying CPUs has not been of a great concern to software developers.

Moreover, since the 1950s when the first compilers appeared [12][13], they have evolved into efficient code converters, bridging the gap between high-level application development and the low-level intricacies of machine instructions. In fact, current

compilers are able to produce compact and speed-efficient codes that rival assembly and machine codes, further widening the gap between high-level application development and low-level hardware intricacies, and making it easier to develop software applications. As such, the number of software developers and software applications has increased drastically over the years.

However, while the FPGA presents an opportunity for higher performance compared to CPUs, these software developers are denied access to this immense capability; therefore, slowing down the adoption of FPGAs as general or accelerated computing platforms. If FPGAs are to become serious contenders as computing platforms, a similar mechanism to compilers, with supporting architectural frameworks must be provided for high-level hardware application development.

Meanwhile, the idea that software developers should adopt HDLs and code directly for hardware is a good proposition. However, this is not a simple solution. Though, in general, HDLs are used to describe the high-level behaviour of circuits at different levels of abstraction – *gate-level*, *register-transfer level* (RTL), *structural*, and *behavioural*, the coding concepts are quite different from software coding. HDLs have statements and constructs that are similar to those in high-level languages for software development, yet they are used to describe intrinsically concurrent behaviours of hardware circuits. This is often a source of confusion to software developers embracing hardware code development. The tendency to think sequentially when dealing with concurrent logic is a major pitfall. Moreover, there are often standard or recommended ways of describing most circuits, and if not followed, this could result in slow, bloated, and power-hungry designs, defeating the purpose of using HDLs in the first place. This means an understanding of the low-level hardware is still required, especially for efficient coding.

Therefore, the ability to develop an application in a high-level language like C and have it automatically compiled to behavioural, structural, or RTL-level HDL codes is an attractive proposition. This is called *High-Level Synthesis* (HLS). In fact, several commercial products [14] in this regard have already been developed with the Xilinx's Vivado HLS [15] being a typical example. This is in the foreground of a substantial

body of academic research [16][17] advancing the underlying techniques and algorithms.

Despite their promise, HLS compilers have limitations [18] as it is in general challenging to produce resource-efficient and power-efficient codes that could rival what an HDL designer would write. However, considering the precedential success of software compilers, given time, HLS compilers will catch up.

Meanwhile, whereas the development of HLS compilers is a welcome advancement, there is still the need for an intermediate architectural model and hardware abstraction layer for computing on the FPGA. Modern FPGAs have immense capabilities that can be better exploited by an intermediate-level abstraction between the output of an HLS engine or a directly coded design and the low-level resources of the FPGA.

While the HLS is a move to address the difficulties of FPGA-based application development, the management of the deployment and runtime execution of applications is still largely unresolved. For instance, Vivado HLS enforces a standard interface for the generated RTL design. However, if the design is meant to be part of a hierarchical design, a separate structural-level HDL coding would still be required to integrate the generated code into an overall design before deployment on an FPGA. Accomplishing this would still present a fairly complicated task to the uninitiated in the art of HDL, further highlighting the need for a task management framework that could automatically abstract task interfacing and inter-task communication, among others.

The reconfigurable hardware fabric is usually under the management of a host CPU for accelerating software processes. With the increased density of modern FPGAs, many more tasks are required to be running simultaneously and often requiring chip area allocation, circuit loading, and so on. In order to free up the host CPU and its resident software operating system from managing these hardware-level functionalities, the *Reconfigurable Operating System* (ROS) has been proposed with early examples including [19] and [20]. The “reconfigurable” terminology derives from the fact that the function of the ROS is to manage the intricacies arising from the reconfigurability of the underlying hardware fabric. Runtime task management needs to be addresses as in fact, a major barrier to the adoption of partial runtime reconfiguration in industrial

applications seems to be the lack of tools and methodologies for runtime task management [21].

The key requirements or services needed in RC have been categorized as task loading, partitioning or floor planning, memory management, scheduling, placement, security, I/O, on-chip communication, and synchronization [22]–[24]. The provision of these services has been generally investigated, both within the context of a ROS and as standalone research efforts. However, without a holistic approach to ROS design that considers multiple but individual design objectives such as *reliability*, *availability*, *real-time processing*, *efficiency*, high *performance*, and *security* in the provision of the RC services, the FPGA could fail to realize its potential as a veritable computing device even while these requirements. It is clear that the afore-mentioned RC services need to meet high standards, the importance of which cannot be overemphasized. It would be interesting therefore, to consider why the need to meet high levels of RC requirements is paramount in current high-end applications.

1.1.1 Reliability is Important

Electronic devices deployed for operation in critical application domains like space, aviation, military, nuclear decommissioning, and nuclear waste management are exposed to extremes of environmental conditions like radiations, temperatures, electromagnetic interferences, and electric fields. These extreme conditions can trigger system-crippling temporary errors and permanent damages in on-board electronics. Reliability is all the more important when human lives are involved, as the failure of electronic devices can result in catastrophic system failures which could lead to disasters with great losses in human lives or economy. Scenarios in the past showed that space missions could go catastrophically disastrous due to the failure of electronic systems [25]. As such, a high premium is placed on device reliability in critical application domains like space and FPGAs targeted at such applications should be able to meet expected levels of reliability. As a further example, radiation-induced errors are taken seriously in space missions as was seen in the Europa Orbiter mission that was replaced with the Europa Clipper Mission for concerns of radiation [26]. With the

increasing use of FPGAs in space applications as evidenced in [27], the tolerance of FPGAs and the designs on them to radiations is important.

Ageing-related errors are also a source of concern especially in space missions, which can last for very long times as seen in the Voyager I and Voyager II space probes which, launched in 1977, have been travelling in space for over 40 years [28]. In contrast, electronics' lifetime has been reduced due to the use of state-of-the-art technologies that push manufacturing processes to the limits. Voyager is a good example of what was possible to achieve using the old and more robust technology, which is extremely constrained in terms of computation power. To achieve something similar using state-of-the-art high-performance COTS technologies, new techniques are needed.

While wireless communication anywhere within the vicinity of earth has appeared almost instantaneous, exploration of deep space has highlighted the limited speed of our wireless communication technology and for now, it does not appear that much can be done in the way of changing that, except if a means of communication other than electromagnetic waves is discovered. Due to the astronomically high distances between cosmic bodies, communication signals take considerable amounts of time to traverse the large expanse of space. As a result, an electronic device deployed in a system ready to explore deep space, for instance, should ideally be able make its own decisions as errors emerge. This calls for a high degree of autonomy and adaptation in such devices.

Meanwhile, because of their inherent high performance capability, FPGAs now find use in critical applications. As a result, reliability techniques that will ensure the autonomous and continuous operation of FPGA-based computing platforms in the face of emergent errors are highly sought after. There are at least two existing solutions to the reliability problem – the use of redundancy at the device level [29], and the use of specialized radiation-hardened (rad-hard) devices [30]. The disadvantage of the former is that it *increases cost and weight*, whereas the rad-hard devices are generally *very expensive* [31], often many orders of magnitude *more expensive* than their COTS equivalents are. Moreover, redundancy at the hardware level can be limited in improving reliability [32] and also have counterproductive effects by adding

complexity and potential failure modes to the system, increasing design opacity, encouraging risk, and discouraging further optimization of individual devices [33].

As such, it is important to consider reliability at the system component level. This work will not only present infrastructures and frameworks for achieving a better overall system reliability, it will also underscore a fault-tolerant implementation of these infrastructures, where applicable.

1.1.2 Availability in High-End Applications

Another factor closely related to reliability is *availability*. The availability of a system is defined in terms of its ability to perform the intended function when required; that is, the amount of time it is in a functional state. On the other hand, reliability measures how well the system has kept to producing the expected output within a given time period. While higher reliability leads to higher availability, the converse is not true. As such, the availability of an FPGA-based system has to be considered as a separate entity, with proper design considerations put in place.

With the proliferation of cloud computing in recent years, it was only a matter of time before FPGAs would be available for computation in the cloud. One hallmark of datacentre and cloud computing is availability [34][35], with other requirements like reliability, and security [36]. For an FPGA in the cloud, these requirements must be met. Downtimes have to be kept to a minimum to maintain a high quality of service since one of the major concerns of organizations with cloud services is assurance of availability or lack of it [34].

Dynamic or runtime programmability will be required to keep an FPGA-based system online while making changes. Incidentally, modern FPGAs come equipped with reconfiguration capabilities that allow a part of the device to be reconfigured while the other parts remain active [37]. These capabilities can be exploited to ensure that FPGAs operating in the cloud and other computing environments provide the expected high levels of availability. This necessitates that proper attention is paid to module-level events that could hamper system-wide availability. This is part of the focus of this work as will be exemplified by the monitoring of errors during task reconfiguration (see Section 4.4).

1.1.3 The Need for Real-Time Computing

Real-time and reliable computing often go hand-in-hand. Because of the stringent requirements of reliable systems, they are often required to be predictable and to meet specified timing requirements since delayed execution of critical tasks can be catastrophic [38]. To ensure these requirements are met, FPGA computing models have to be such that task execution deadlines are met. If access to shared resources is not well managed, this could result in missed execution deadlines, a situation that is particularly undesirable in real-time systems. In particular, the internal configuration interface of current FPGAs is singular and is required for multiple system-level functionalities like hardware task configuration and error mitigation. In order to ensure execution deadlines are met, every system function including their overheads must have predictable behaviour and bounded timing.

In this research, real-time factors are put into consideration where necessary and applicable. For instance, the on-chip communication scheme proposed is implemented to ensure guaranteed latencies. In addition, a mechanism is provided for a more efficient management of the internal configuration interface for multiple complimenting system functions.

1.1.4 The Importance of Security

The adoption and support of FPGAs by technology giants like Microsoft, Amazon, and Baidu represents a step change in the prospect of deploying FPGAs as computing devices. Recently, Microsoft started using FPGAs in their Bing search engine servers with almost two-fold improvement in search ranking throughput at only 10% increase in power consumption [3][39]. In addition, Amazon has introduced cloud-based pay-per-use FPGA instances and an IP market place, providing ready access to a platform for accelerated cloud computing [4]. Baidu, on the other hand, uses FPGAs to accelerate large-scale deep neural networks and online services at low power cost [40][41]. These developments have served to increase interests in FPGAs and the value of IP cores that run on them. IP core vendors pay heavily in monetary terms and development time to design these IP cores. As a result, the protection of this investment is of paramount importance. However, malicious attacks have aimed to exploit the security

vulnerabilities of FPGAs to steal these IPs or cause undesirable effects. Meanwhile, as reported in [24], the issue of security has been given very little attention in reconfigurable computing. As such, another subject in this work is the *investigation of efficient techniques for secure task reconfiguration*.

1.2 Thesis Scope and Objectives

The scope of this thesis is represented diagrammatically in Figure 1.1. The broad objective is to deliver the enabling frameworks for *reliable*, *available*, *real-time*, *efficient*, *secure*, and *high-performance* reconfigurable computing by providing techniques, methods, and mechanisms for runtime *reconfiguration*, and dynamic inter-circuit *communication* and synchronization for circuits on reconfigurable hardware.

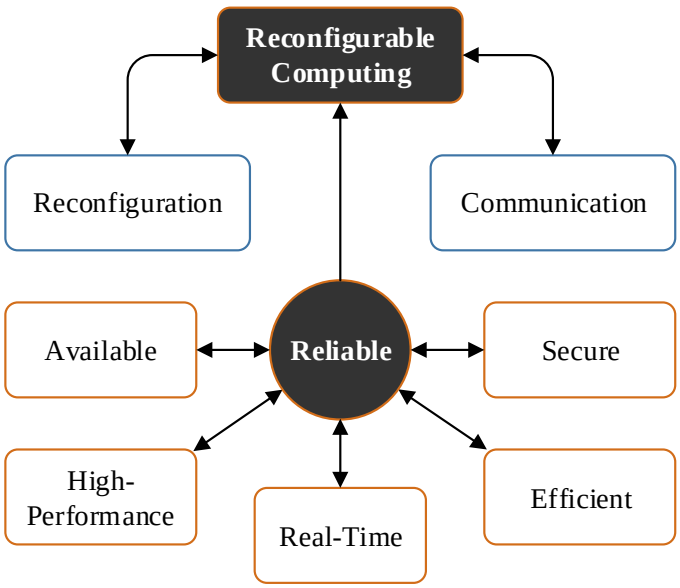


Figure 1.1: Diagrammatic representation of the scope of this thesis

The aim is for the concepts advanced in this work to serve as the bedrock for a computing paradigm where a COTS reconfigurable hardware like an FPGA is used as a standalone or co-processing heterogeneous computing platform for *reliable* dynamic task management, as opposed to using very expensive rad-hard devices. As Figure 1.1 shows, reliability is a feature theme and all the other features are implemented in the light of this. In particular, the specific aims of this research include:

- The development of a high-performance and efficient configuration engine for reliability and high availability in reconfigurable systems,
- The investigation of reconfiguration mechanisms for secure bitstreams, and
- The development of a dynamic network-on-chip infrastructure for intrinsic reliability-aware inter-task communication and synchronization.

While the specific applications of the novel methods described in this thesis have been implemented and prototyped on FPGAs, the ideas and concepts are generic enough for application in other reconfigurable hardware and even in custom computing devices.

1.3 Contribution to Knowledge

The contributions of this research stem from the innovative approaches applied in the realization of the specific aims of this work. While the deliverables of this work are targeted at reconfigurable computing, applications in closely related fields would be anticipated. The major contributions of this work are:

- A novel configuration mechanism for efficient runtime task loading, configuration memory readback, circuit and data relocation, and configuration error monitoring
- An efficient method for soft error mitigation that explores a criticality-aware selective-area scanning of the chip for soft error detection and correction
- A new and unique on-chip inter-circuit communication and synchronization technique with intrinsic support for circuit relocation, which promises to be an enabler for an improved system-level reliability in COTS reconfigurable hardware devices
- A highly efficient mechanism for relocating circuits with encrypted configuration bitstreams without the need for on-chip decryption of the bitstream, ensuring that the security of IPs is not compromised nor maintained at a high cost of on-chip resources and system time
- A unique relocation-aware configuration bitstream format for resource- and time-efficient secure circuit relocation

- A unique software interface for relocation-aware configuration bitstream formatting and encryption key reassignment

1.4 Target Device and Development Environment

The target FPGA family in this research is the Xilinx 7 series and all the HDL code implementations are evaluated on a 7 series device, unless stated otherwise. Moreover, when necessary, the Zynq-7000 FPGA, which contains an Arm® Cortex™-A9 processor tightly connected with a 7 series FPGA is used for implementations. Nevertheless, the techniques developed in this work can be easily extended to other FPGA families, including the ones from other vendors. The Vivado Design Suite is used for all synthesis and implementation.

1.5 Thesis Outline

Chapter 2 reviews the techniques for the reconfiguration of hardware devices with particular attention to dynamic reconfiguration, which is the enabler for fine- and coarse-grained access to on-chip resources in runtime. Moreover, since FPGAs are becoming increasingly used as computing platforms, the security implications of this are great. Chapter 2 provides an insight into the provision made by vendors to ensure the security and integrity of reconfigurable devices and the applications that run on them.

In Chapter 3, the concept of reconfigurable computing is introduced. Attention is drawn to the reliability concerns in reconfigurable hardware and existing mitigation approaches discussed. A key concept to look out for is partial bitstream relocation, which is important for reliability. A discussion of its requirements is presented as well as the existing body of work which has provided techniques for meeting these requirements. A foray is taken into the issues of configuration and communication in RC systems and existing approaches are examined. As real-time processing is closely related to reliability, real-time system requirements are studied and the implications of these on configuration and communication are brought to light.

A configuration controller with a unique set of reliability-enabling features is crucial for RC. In Chapter 4, the design and implementation of a high-performance configuration memory access controller are presented. Key RC-supporting functionalities for enabling a reliable ROS are advanced. These include task configuration and relocation to circumvent permanent damages, configuration memory readback, internal register readback for device diagnosis, and configuration monitoring and abort mechanisms for device availability improvement. Partial bitstream relocation capability is implemented intrinsically by treating every configuration operation as a relocation request In addition, an error mitigation functionality that allows selective-chip-area scanning and thus saves on time, is put forward.

To the best of the author's knowledge, the concept of relocating encrypted partial bitstreams has not been met with a unifying approach. Methods already exist that can be used but this can at best be seen as an amalgamation of several distinct methods, resulting in bloated time and resource utilization. Chapter 5 proposes a novel method for achieving this with virtually no time and resource overhead when compared to existing methods. In order to achieve this, a new and unique bitstream format is proposed for encrypted partial bitstreams and a corresponding configuration controller implemented to load a bitstream so formatted. The algorithm is further implemented in a Windows-Form-based application to provide an easy-to-use graphical user interface.

A key requirement for partial bitstream relocation is the provision of dynamic communication framework as circuits are relocated in runtime. An on-chip communication mechanism with intrinsic relocation support will therefore provide a step change in the reliability of reconfigurable systems. A move is taken in this direction in Chapter 6 by proposing a communication framework that is amenable to circuit relocation. The clock buffers and nets of the FPGA are adapted for communication. Being a new concept that has never before been investigated (as far as the author is aware), salient technical considerations and characterizations are presented. The chapter starts with an introduction to clocking resources in a typical FPGA and highlights the features exploited for communication. Different configurations of clock buffers to enable a variety of network topologies are also investigated. A demonstration of bitstream relocation that derives communication support from clock buffers is as

well presented. The chapter concludes with the implementations for a fault-tolerant data transfer via the clock buffers.

In Chapter 7, the frameworks proposed in this work are evaluated with a practical application. A case study is drawn from the NASA/JPL spectrometer application. The data processing circuits of this spectrometer are used as hardware tasks and the performance of the methods in this work are compared with those of existing approaches.

This thesis concludes with a retrospective look at the aims and objectives of this work. In particular, it discusses how these have been met in the light of the methods and techniques advanced in this work. Limitations and the possible route to addressing them are also furnished, especially in relation to the limitations of the target device and improvements offered by the latest devices from vendors. In addition, future directions for the work are presented. Specifically, this mostly involves how all the contributions in Chapter 4 through Chapter 6 tie together and enables a complete high-performance and reliable RC system.

1.6 Relevant Publications

Journals

- **A. Adetomi**, G. Enemali, and T. Arslan, 'Enabling Dynamic Communication for Runtime Circuit Relocation', *IEEE Transactions on Very Large Scale Integration Systems*. [Submitted].
- **A. Adetomi** and T. Arslan, 'A High-Throughput and Efficient Configuration Controller for Reliability in Dynamically Reconfigurable Systems', *IEEE Transactions on Computers*. [Submitted].
- **A. Adetomi** and T. Arslan, 'A Security-Aware Relocation Mechanism for Reliable Reconfigurable Computing', *IEEE Transactions on Parallel and Distributed Systems*. [Submitted].
- G. Enemali, **A. Adetomi**, G. Seetharaman, and T. Arslan, 'A Functionality-Based Runtime Relocation System for Circuits on Heterogeneous FPGAs', *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2018, vol. 65, no. 5, pp. 612–616.

Conferences

- **A. Adetomi**, G. Enemali, Xabier Iturbe, Didier Keymeulen, and T. Arslan, 'R3TOS-Based Integrated Modular Space Avionics for On-Board Real-Time Data Processing', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 1–8.
- **A. Adetomi**, G. Enemali, G. Seetharaman, and T. Arslan, 'Fault-Tolerant Mechanisms for Relocation-Aware Dynamic On-Chip Communication on FPGAs', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 214–217.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Towards a Secure Partial Reconfiguration of Xilinx FPGAs', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 174–178.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Characterization of Clock Buffers for On-Chip Inter-Circuit Communication in Xilinx FPGAs', in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Towards an Efficient Intellectual Property Protection in Dynamically Reconfigurable FPGAs', in *2017 Seventh International Conference on Emerging Security Technologies (EST)*, 2017, pp. 150–156.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Relocation-Aware Communication Network for Circuits on Xilinx FPGAs', in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'A Fault-Tolerant ICAP Controller with a Selective-Area Soft Error Mitigation Engine', in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 192–199.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Relocating Encrypted Partial Bitstreams by Advance Task Address Loading', in *25th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2017)*, 2017, pp. 188–191.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Clock Buffers, Nets, and Trees for On-Chip Communication: A Novel Network Access Technique in FPGAs', in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 219–222.
- G. Enemali, **A. Adetomi**, and T. Arslan, 'Efficient Runtime Frame ECC Recomputation for Reliable Task Execution on Xilinx FPGAs', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 59–65.
- Aliyu Dala, **A. Adetomi**, G. Enemali, and T. Arslan, 'RR4DSN: Reconfigurable Receiver for Deepwater Sensor Nodes', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 280–284.
- G. Enemali, **A. Adetomi**, and T. Arslan, 'A Placement Management Circuit for Efficient Realtime Hardware Reuse on FPGAs Targeting Reliable Autonomous

Systems', in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

- G. Enemali, **A. Adetomi**, and T. Arslan, 'Expanding the Un-usable Area Strategy for Improved Utilization of Reconfigurable FPGAs', in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 139–144.
- W. Guohua, L. Dongming, W. Fengzhou, **A. Adetomi**, and T. Arslan, 'A Tiny and Multifunctional ICAP Controller for Dynamic Partial Reconfiguration System', in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 71–76.
- G. Enemali, **A. Adetomi**, and T. Arslan, 'FAReP: Fragmentation-Aware Replacement Policy for Task Reuse on Reconfigurable FPGAs', in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 202–206.
- J. Khalifat, A. Ebrahim, **A. Adetomi**, and T. Arslan, 'A Dynamic Partial Reconfiguration Design for Camera Systems', in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2015, pp. 1–7.

Chapter 2

2

Reconfiguration, Security, and Integrity in FPGAs

The design on an FPGA is mapped by a *Configuration Memory* (CMEM), which holds information about the functional state of all the resources on the chip, including routings. Depending on the type of memory technology used for the CMEM, an FPGA can be categorized as SRAM (*Static Random Access Memory*), *flash*, or *antifuse*; with the majority of FPGAs in the market being SRAM-based, including the ones from Xilinx, which is considered as the inventor of the FPGA [10] and holds the largest market share [42].

Since the SRAM memory is volatile, SRAM FPGAs need to be reprogrammed each time the device is power-cycled and an external non-volatile memory is used for the application program's storage. Any digital design implemented on an FPGA uses the reconfigurable logic and memory resources provided by the FPGA, interconnected together by means of routing resources. When synthesizing the digital design, a *bitstream* is generated that contains the configuration information. In order to physically realize the digital design on the FPGA this bitstream or bit file is written to the CMEM. Ranging from serial to parallel, and external self-reconfiguration to internal partial reconfiguration, there are several means of loading or reconfiguring an FPGA. The configuration details, interfaces, modes, reconfiguration strategies of FPGA will be explored in Sections 2.1 and 2.2.

The obvious disadvantage of SRAM FPGAs is the perceived greater vulnerability to attacks. Since the configuration bitstream is kept outside the device, the process of transferring it to the FPGA is exposed to the prying eyes of attackers. As a result, SRAM FPGAs are generally considered to be less secure when compared to flash and antifuse FPGAs, which are based on non-volatile memories. However, both flash and antifuse FPGAs have their security challenges as well. The in-system programming

commonly employed in flash FPGAs exposes the FPGA to the same security vulnerabilities as SRAM FPGAs and system-level security concerns may have to be addressed in antifuse FPGAs since they are one-time programmable and the configured digital design cannot be erased [43].

In general, from the security standpoint, all the types of FPGAs have their strengths and weaknesses. However, SRAM FPGAs are ubiquitous because they are often a few process nodes ahead of other technologies [10], with associated benefits of higher performance, enhanced power efficiency, and greater logic density; including being easy to manufacture, test, and update in the field [43].

In all, there are varieties of potential attacks on FPGAs and some countermeasures have been provided by FPGA manufacturers to protect the FPGA and maintain the integrity of the on-board digital circuits. Some of these will be brought to light in Section 2.3.

2.1 FPGA Configuration Details

FPGAs are generally composed of columns and rows of *Configurable Logic Blocks* (CLBs), *Block Random Access Memories* (BRAMs), *Digital Signal Processors* (DSPs), and interconnect resources (wires, connection boxes, and switch boxes). There are also clocking infrastructures, and a host of other primitives for functionalities like internal configuration and analogue-to-digital conversion. Inside a CLB are *Flip-Flops* (FFs), and *Look-Up Tables* (LUTs) for implementing various digital circuits. All these resources can be controlled by accessing an on-chip configuration memory from outside or inside the FPGA.

2.1.1 Layers of an FPGA

For the purpose of clarity, different layers will be identified on the FPGA. Since the CMEM holds information about the functional state of all the resources on the chip, including routings and clock networks, that means an FPGA can be pictured as having a *configuration layer* which determines the behaviour of the *functional* or *application layer*, where the circuit designs reside. The functional layer comprises the user

resources – logic and memory blocks. In addition, the FPGA has a dedicated clock network for clocking synchronous circuits and as such, a third layer, which can be considered as a subset of the functional layer can be identified as the *clock layer* (or *clock routing layer*).

Figure 2.1 shows a pictorial relationship among all the layers – the functional layer for mapping designs on the FPGA's fabric, with a sub-functional-layer for clocking, and the configuration layer for loading the designs into a configuration memory. The clock layer contains dedicated networks for routing clock signals to relevant resources and a number of clock conditioners and buffers for steering signals to these networks [44]. *Programmable Interconnect Points* (PIPs) are provided for both logic and clock routing. The logic routing PIPs are in the *Connection Boxes* (CBs) and *Switch Boxes* (SBs). The PIPs are largely CMOS pass transistors that are also mapped in the CMEM.

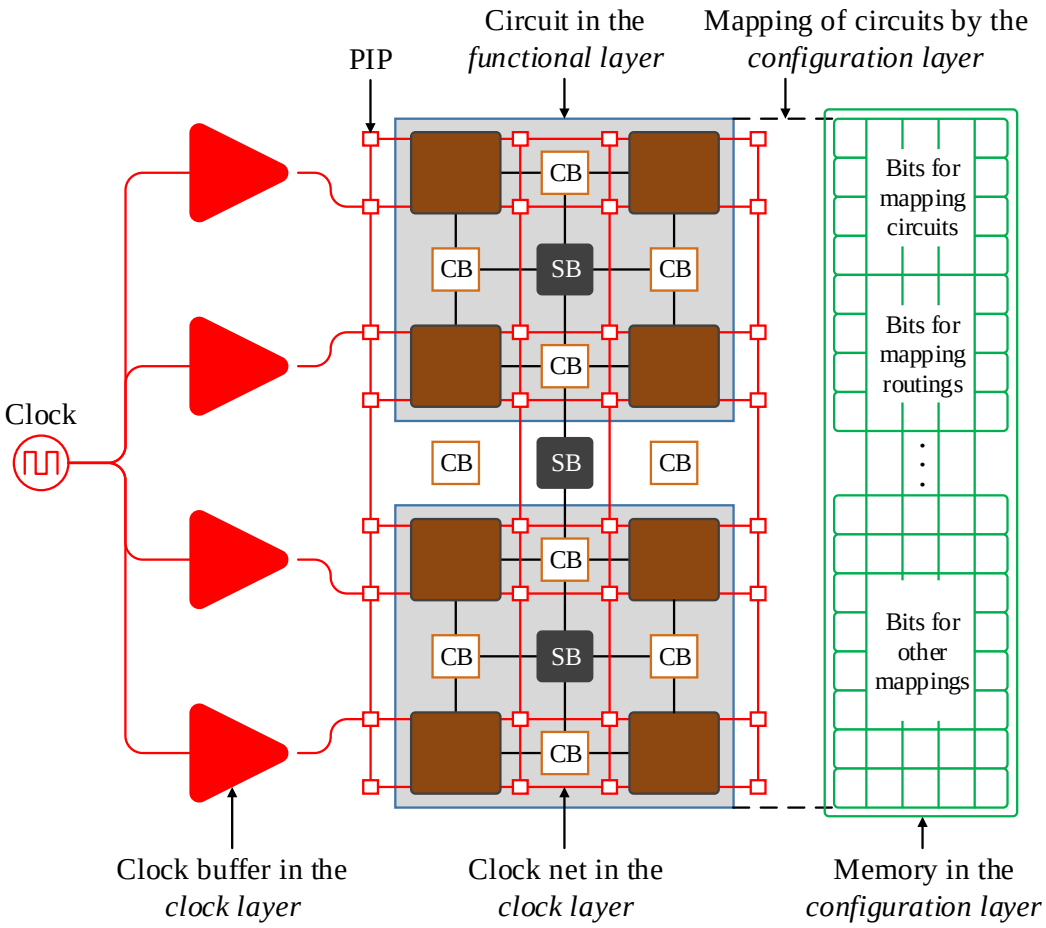


Figure 2.1: Three layers of an FPGA

A. Functional Layer

The FPGA is made up of a reconfigurable hardware fabric with multiplicity of logic elements. Figure 2.2 is the traditional model of an FPGA, showing inactivated routing resources that can be programmed to interconnect configurable logic resources and *Input/Output Blocks* (IOBs). The CBs are used to connect directly between the input and outputs of programmable resource blocks (CLBs, BRAMs, and DSPs) while the SBs are used to switch between vertical and horizontal connections, allowing a wide range of routing options [45]. In the diagram LMD represents logic, memory, or DSP resources. In the 7 series, each CLB comprises of 2 slices and in each slice there are eight flip-flops, four 6-input LUTs, one arithmetic and carry chain, with each LUT configurable as a 64-bit distributed RAM or 32-bit shift register logic [46].

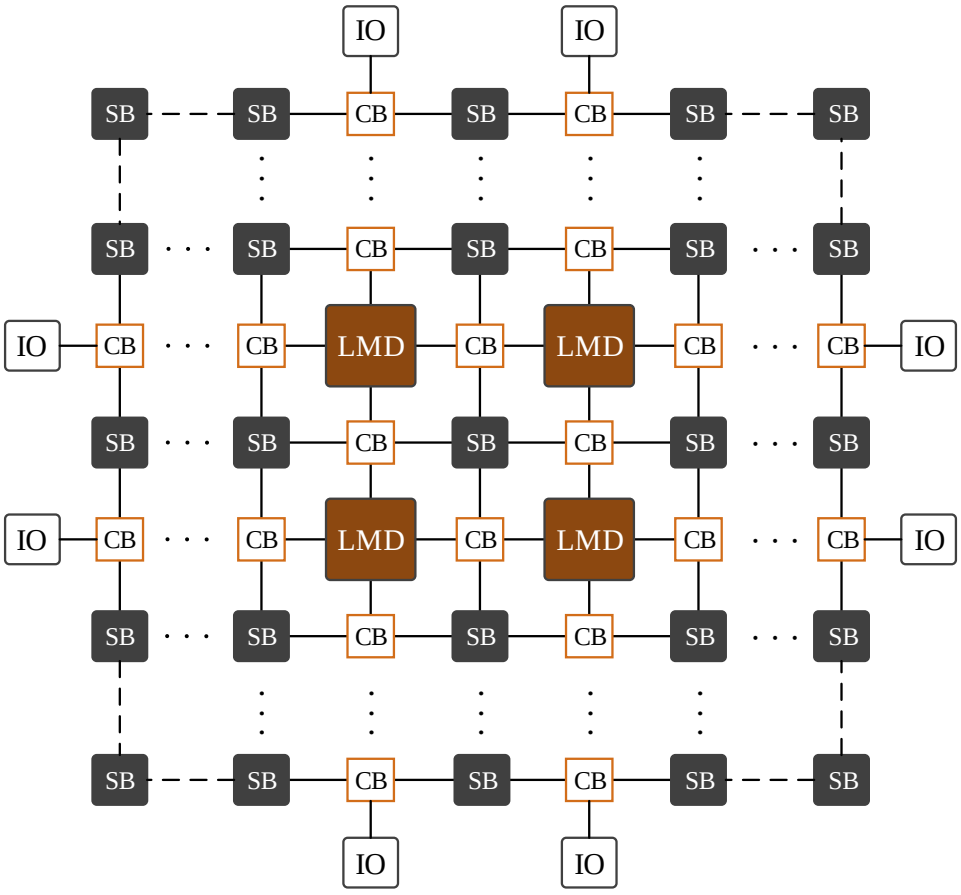


Figure 2.2: Model of a typical FPGA showing the interconnection among on-chip resources and input/output (IO) blocks. LMD represents logic, memory, or DSP resources.

Circuit designs for the FPGA are crafted using the resource blocks and configured on the FPGA by writing a bitstream to the CMEM of the FPGA. Since the 7 series FPGAs (as with the other families) are SRAM-based and as such the CMEM is volatile, the bitstream is always required to be loaded each time the device is power-cycled. In certain instances, FPGAs also come equipped with processing elements or cores like CPUs and GPUs. These are referred to as *hard cores* since they are permanent structures on the chip and are distinct from the reconfigurable fabric. However, sometimes processing elements not provided in fixed silicon by the FPGA can be obtained by using the existing CLBs, BRAMs, and DSPs to implement needed functionalities, as is the case with Xilinx MicroBlaze soft processor. Such cores are referred to as *soft cores* and offer more flexibility compared to their hard counterparts, though generally at a reduced performance.

B. Clock Routing Layer

To drive the clock and reset inputs of synchronous circuits on the FPGA, clocking infrastructures are provided. In the latest FPGAs like the 7 series, the CLBs, BRAMs, and DSPs are arranged in a grid of columns and rows, with contiguous number of same-height columns grouped into *clock regions*. While the size and number of clock regions varies among device families, each clock region is fed by a number of local clock buffers and nets. To cater for the diverse clocking need of the user design, there are also multi-region and global device-wide clock networks spanning different groupings of clock regions. A clock region marks the granularity of the clock network of the FPGA. It is important to note that these clock networks use dedicated physical interconnect resources that are independent of the local and general routing resources. As such, an FPGA can be considered as having a *clock routing layer*, which is in addition to the functional and configuration layers. Nonetheless, the clock networks and their associated infrastructures have bits in the CMEM for controlling them.

In the 7 series, a clock region spans 50 vertical CLBs, 10 36-kb BRAMs, 20 18-kb BRAMs, or 20 DSPs and depending the device size, there can be up to 24 clock regions [44]. Figure 2.3 gives a representation of a typical 7 series FPGA with respect to the location of the clock buffers.

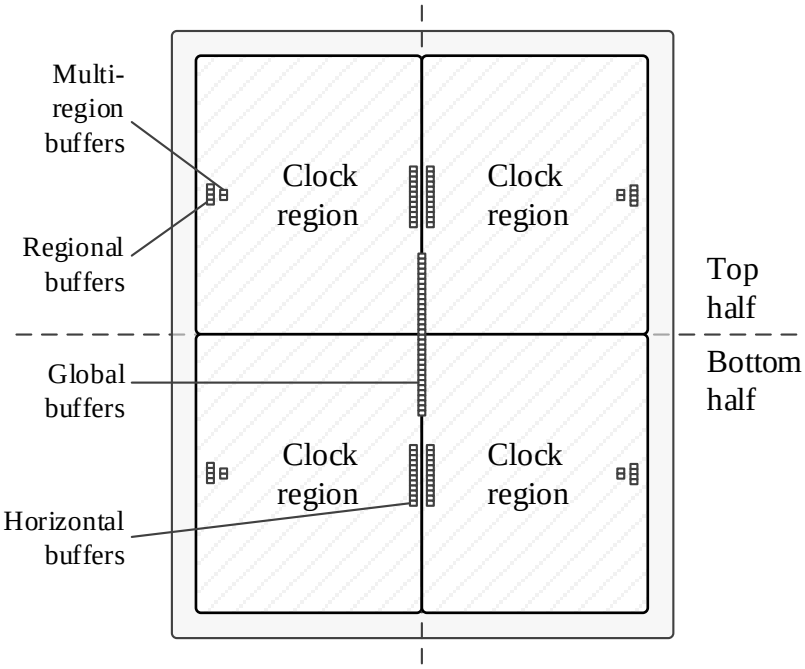


Figure 2.3: Clock buffer distribution in a 7 series FPGA

C. Configuration Layer

The configuration layer includes the CMEM, the configuration access ports and the internal control logic for loading circuits' configuration data into the CMEM. The CMEM holds the functionality (including routing) of circuits mapped into the FPGA. Physically, the CMEM is tiled about the device in configuration frames that run from the top to the bottom of a clock region. In the 7 series FPGA, these frames contain 3,232 bits (101 32-bit words) each [47], with each bit stored in an SRAM cell. The *configuration frame* is the smallest unit of configuration and all CMEM read or write operations are therefore required to access whole configuration frames. It is the smallest addressable segment of the FPGA's CMEM space and depending on the amount and type of resources to be configured, a number of frames have to be written to the FPGA.

The internal structure of the configuration frames resembles the physical arrangement of the FPGA resources in the clock region column (see Figure 2.4). The most significant bits in the frame are associated with the upper resources in the clock region, whereas the least significant frame bits configure resources in the bottom part of the clock region. Likewise, the configuration related to the dedicated regional clock wires that cross through the central part of clock regions are mapped to the middle 51st

word in the frame. The configuration information for a given FPGA resource spans along several consecutive frames, always occupying the same relative position within them. For instance, the configuration information for one single CLB is spread along 36 frames. Similarly, 128 frames are required to configure a BRAM, and 28 frames for a DSP slice (see Table 2.1). The third row in Table 2.1 gives the number of words in each frame that configures a single resource block. These numbers can be obtained by studying the bitstreams generated for circuits with different combinations of resource blocks.

Each configuration frame is protected with 13 *Error Correction Code* (ECC) bits that are also embedded in the 51st word. These ECC bits allow for detecting and correcting single bit errors and detecting but not correcting double bit errors (see Section 2.3.2).

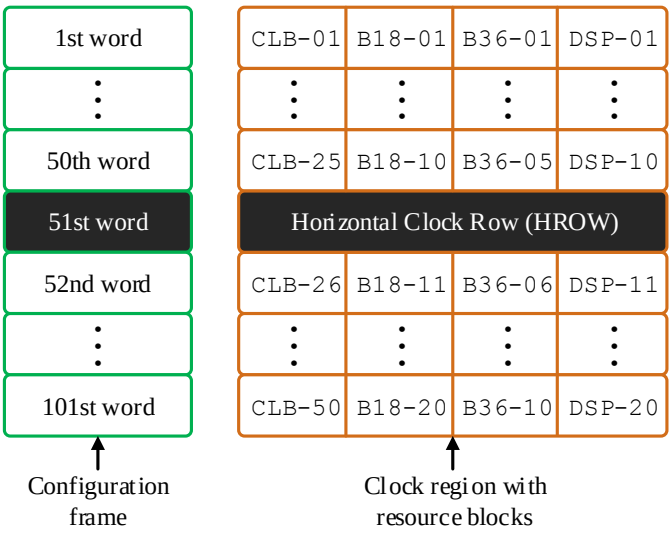


Figure 2.4: Internal composition of a frame and its mapping to resources

Table 2.1: Number of configuration frames for 7 series FPGA resources

Resource Block	CLB	DSP	BRAM18	BRAM36
Number of Frames/Column	36	28	128	128
Number of Words/Block	2	5	5	10

The configuration frame data in the CMEM are addressed by a 32-bit address space. For accessing frame data, a frame address has to be written into the *Frame Address Register* (FAR). Since there are many frames to be configured for the resource blocks, the FAR register is usually loaded with the starting frame address. The internal configuration logic automatically increments the frame address as new frames are loaded. Table 2.2 (adapted from [47]) shows the fields of a frame address. Unique codes are used to identify different resource blocks.

Table 2.2: Frame address fields and corresponding description

Field	**Bit Index**	**Description**
Block Type	[25:23]	Valid block types are CLB, I/O, CLK (000), block RAM content (001), and CFG_CLB (010)
Top/Bottom Bit	[22]	Select between top-half rows (0) and bottom-half rows (1)
Row Address	[21:17]	Selects the current row. The row addresses increment from centre to top and then reset and increment from centre to bottom
Column Address	[16:7]	Selects a major column, such as a column of CLBs. Column addresses start at 0 on the left and increase to the right
Minor Address	[6:0]	Selects a frame within a major column

Table 2.3 shows selected registers of the 7 series FPGA. Some of these registers have read/write capabilities while others have only read or write access. Each register write takes two words – one for the command and the other for the register value, except for the FDRI register which receives a multiple of 101 words as frame data. From the standpoint of CMEM access, the most important are the FAR, CMD, FDRI, and FDRO registers. The CMD register is used to specify what type of operation the FPGA should perform. Table 2.4 presents the CMD register commands and codes.

For every write to the FDRI register, the identity of the device must be specified by writing to the IDCODE register, and a pre-computed CRC checksum can be written to the CRC register at the end of data loading to verify the integrity of the bitstream loaded. To reset CRC check prior to the loading of data, the RCRC command is used. The FPGA also exposes registers for device setup, status, and other device-level functionalities, e.g., CTL0, STAT and COR0 registers.

Table 2.3: Selected configuration registers of the 7 series FPGA

Name	R/W	Address (Binary)	Description
CRC	R/W	00000	CRC Register
FAR	R/W	00001	Frame Address Register
FDRI	W	00010	Frame Data Register Input (for writing configuration data)
FDRO	R	00011	Frame Data Register Output (for reading configuration data)
CMD	R/W	00100	Command Register
CTL0	R/W	00101	Control Register 0
MASK	R/W	00110	Masking Register for CTL0 and CTL1
STAT	R	00111	Status Register
COR0	R/W	01001	Configuration Option Register 0
MFWR	W	01010	Multiple Frame Write Register
CBC	W	01011	Initial CBC Value Register
IDCODE	R/W	01100	Device ID Register

R = Read, W = Write

Table 2.4: Selected CMD register commands and codes

Command	Code	Description
NULL	00000	Null command, does nothing
WCFG	00001	*Writes Configuration Data*: used prior to writing configuration data to the FDRI
MFW	00010	*Multiple Frame Write*: used to perform a write of a single frame data to multiple frame addresses
RCFG	00100	*Reads Configuration Data*: used prior to reading configuration data from the FDRO
START	00101	*Begins the Startup Sequence*: The startup sequence begins after a successful CRC check and a DESYNC command are performed.
RCRC	00111	*Resets CRC*: Resets the CRC register
DESYNC	01101	*Resets the DALIGN signal*: Used at the end of configuration to desynchronize the device. After desynchronization, all values on the configuration data pins are ignored

Two main operations can be performed on the CMEM – *configuration* (writing), and *readback* (reading). By writing to the CMD and FAR registers, and reading from the FDRI register, which holds the frame data read back from the CMEM, a readback

operation can be performed. Similarly, for a configuration operation, the CMD, FAR, and FDRO registers must be accessed. More details on the configuration details can be found in [47].

2.1.2 Bitstream Structure

In order to configure the on-chip resources and control device-related options, the configuration registers of the FPGA are accessed using variable length command packets (see Table 2.5). The minimum size of a packet is 64 bits and the FPGA is equipped with an internal 64-bit packet buffer. From this point forward unless otherwise stated, a word is to be taken as 32-bit long. The first 32 bits of the packet are a Type 1 header, which defines the type of register operation, the register address and the word count (number of words to read or write). If the word count is less than 2048, there is a second 32-bit header which redefines the register operation and specifies the word count. If a Type 2 header has to be used a Type 1 header must be specified first although with the Type 1 header word count set to zero. When writing a register, the packet includes a variable-length register data word, the length depending on the register being written. For instance, when writing the FDRI register, a multiple of 101 words must be provided.

Table 2.5: Packet format for configuration command and data

Field	***Header***		***Body***
Content	Type 1 header	Type 2 header	Register data words
Length	32 bits	32 bits	Variable
Description	This header is always used	*Optional*: Used only if the word count is > 2047	This field is needed only when writing a register. When writing the FDRI register, a multiple of 101 words must be loaded

The Type 1 packets can read or write up to 2047 words while the Type 2 packets can access larger blocks of up to $(2^{27} – 1)$ words [47]. Table 2.6 and Table 2.7 show the composition of the Type 1 and 2 packets respectively, with read and write operation codes indicated in Table 2.8.

Table 2.6: Type 1 packet header format for configuration commands

Header Type	Opcode	Register Address	Reserved	Word Count
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	xx	RRRRRRRRRxxxxx	RR	xxxxxxxxxxx

Table 2.7: Type 2 packet header format for configuration commands

Header Type	Opcode	Word Count
[31:29]	[28:27]	[26:0]
010	xx	xxxxxxxxxxxxxxxxxxxxxxxxxxx

Table 2.8: Opcode format for configuration commands

Function	NOP	Read	Write	Reserved
Opcode	00	01	10	11

By using the register addresses in Table 2.3 with the Types 1 and 2 packet headers, the configuration commands can be formed. Table 2.9 gives an example of the formation of the command packet to write the CMD register. This packet is followed immediately by a command code. For example, to write configuration data, one would write 0x30008001 followed by 0x00000001, which is the WCFG code (see Table 2.4) before loading the data.

Table 2.9: Forming the packet header to write the CMD register

Packet Field	Write CMD	Description
Header Type [31:29]	0b001	Header type 1
Opcode [28:27]	0b10	Write
Register Address [26:13]	0b00000000000100	Register address “00100”
Reserved [12:11]	0b00	Reserved bits, set to 0s
Word Count [10:0]	0b00000000001	Write one word
Packet [31:0]	**0x30008001**	Full packet in hexadecimal

A bitstream eventually, is composed of many packets concatenated into a single bit file and used to program the FPGA. In general, a bitstream can be divided into multiple sections of *configuration commands* and *frame data*. The configuration commands are the packets that set device options and control the configuration logic of the FPGA, including writing the FAR register. The frame data are the register data words written to the FDRI register in order to set the functional state of the resources on the FPGA including controlling general interconnect resources and clock networks. The frame data is what gets written to the CMEM.

Figure 2.5 shows the general format of the Xilinx bitstream. The bitstream includes a *preamble*, a *body*, and a *postamble*. The preamble includes special words for synchronizing the configuration logic and header command words for setting up the FPGA. The body is the bulk of the bitstream as it contains FAR loading commands, FAR values (starting frame addresses for the different resource blocks used by the design), and frame data. The postamble contains commands for starting up the FPGA after frame data loading. It also contains the CRC checksum and the DESYNC command packet (0x3000800100000000D) for desynchronizing the configuration interface. The bitstream concludes with a couple of no-operation words (0x20000000) for flushing the internal packet buffer.

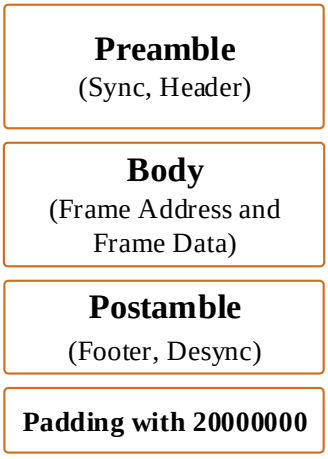


Figure 2.5: Xilinx bitstream format

Xilinx provides Vivado *Integrated Development Environment* (IDE) [48] for simulating, debugging, synthesizing, and implementing designs for their FPGAs with various design flows, including the generation of bitstreams. The Hardware Manager, a feature in Vivado can be used be for externally programming FPGAs.

2.1.3 Configuration Interfaces and Modes

The Xilinx 7 series FPGA offers different means of (re)configuring the chip (see Table 2.10, adapted from [47]), with varying configuration port interface modes, widths and bandwidths. It should be noted however, that only one of these interfaces can access the internal configuration logic at a time, with the *Joint-Test Action Group* (JTAG) interface having the highest priority. Some of the configuration interfaces are serial, some are parallel, while others offer a combination of both serial and parallel. The interfaces in master mode can drive the clock of the configuration logic from an internally-generated clock at a maximum frequency of 100 MHz [49], though the Vivado IDE allows a designer to specify only a maximum of 66 MHz when using the internal oscillator option (master interface) for configuration clock generation [50]. However, for higher speeds, the slave modes can be used with a configuration clock sourced externally.

Table 2.10: Configuration interfaces and modes in the Xilinx 7 series FPGA

Interface	**Mode**	**Bus Width**	**Clock Source**
Serial	Master Serial	x1	Internal
	Slave Serial	x1	External
SPI	Master SPI	x1, x2, x4	Internal
BPI	Master BPI	x8, x16	Internal
SelectMAP	Master SelectMAP	x8, x16	Internal
	Slave SelectMAP	x8, x16, x32	External
JTAG	-	x1	Not Applicable

From the consideration of reconfigurable computing, it is necessary to have an internal configuration interface or mode that allows an FPGA-based system to be self-

contained, without requiring an external processor when necessary. While the *Serial Peripheral Interface* (SPI) and *Byte Peripheral Interface* (BPI) allow the FPGA to reconfigure itself by reading the bitstream out of an SPI or BPI flash memory, this is only applicable to full configuration. Only the JTAG, Serial, and SelectMAP interfaces support partial reconfiguration, with the SelectMAP in addition providing an *Internal Configuration Access Port* (ICAP) that a user design can use to access the configuration memory from inside the FPGA; providing self-reconfiguration capability without the need of external pins.

The ICAP (named ICAPE2 in the 7 series FPGA) primitive is in essence, an internal version of the SelectMAP interface and it grants the user design access to the configuration memory of the FPGA. The diagram in Figure 2.6 shows the ICAP interface in the 7 series FPGA. CSI_B is the active-low ICAP enable port while RDWR_B is the read/write select line. The main difference between the ICAP in the 7 series and the earlier FPGA family (Virtex-6) is that for readback, there is no longer a BUSY port to monitor in order to know when readback data on the output port is valid. In the 7 series, readback validity is deterministic, with data valid 3 clock cycles after CSI_B is asserted. The maximum bandwidth of the ICAP is 400 MB/s at the recommended maximum operating frequency of 100 MHz.

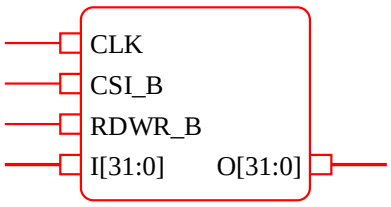


Figure 2.6: 7 series FPGA's ICAP interface ports

2.2 Reconfiguration Strategies in FPGAs

In terms of the granularity of reconfiguration, FPGAs started out as monolithic reconfigurable devices where the entire device had to be taken offline to reprogramme it, significantly adding to downtime. Since then, FPGAs have come a long way as per their capability to be reconfigured. The following subsections provide an overview of

the available means of (re)programming FPGAs, setting a context for the need for dynamic task reconfiguration in a ROS.

2.2.1 Full Reconfiguration

With a full reconfiguration, the entire FPGA is held in reset while the full bitstream is loaded into the device. This is the classic method of programming FPGAs, and one which is limiting in applications that require a high uptime, as it turns out that even for a small change to the design on the FPGA, the entire device has to be taken offline for full reconfiguration. To generate a bitstream for full reconfiguration, the steps in Figure 2.7 are generally followed.

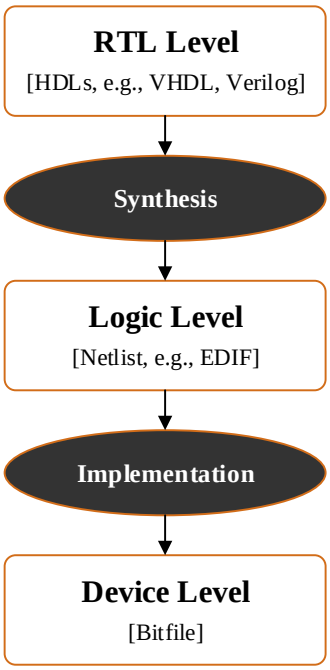


Figure 2.7: Typical FPGA design flow, from synthesis to bitstream generation

According to Figure 2.7, the design is coded up in an HDL language like VHDL or Verilog. This is at the RTL level, with behavioural or gate-level hardware descriptions. The synthesis process converts RTL codes into a netlist descriptions, which can be implemented into bit files (bitstreams) for loading onto an FPGA. The process of implementation includes the placement of logic blocks, memory elements, and other design elements in specific sites on the chip as defined by the netlist. It

concludes with the establishment of routes to interconnect these elements, also as defined by the netlist. The last stage is the generation of bitstream. A more comprehensive FPGA design flow would include simulation and design verification to ensure the correctness of the design both logically and functionally [51].

2.2.2 Partial Reconfiguration

Partial reconfiguration (PR) is a technology that allows a part of an FPGA to be reconfigured while the other parts retain their configured state [37]. The region that is not marked out for PR remains static while other regions can be partially reconfigured. Unlike a full reconfiguration, the entire device is not reconfigured, and as such, PR has advantages such as reduced device size, lower device count, and reduced cost arising from the fact that many circuits can be time-multiplexed to share a single region on the FPGA. Other benefits include in-field local or remote hardware servicing and updating, shorter reconfiguration times, and increased system performance (lower downtime) [52].

There are two types of PR: static partial reconfiguration, and *Dynamic Partial Reconfiguration* (DPR). In the former, the remaining part of the FPGA is brought into a shutdown state – the circuits in the region not being reconfigured stop operating but the corresponding CMEM content is not reset. This is also referred to as shutdown partial reconfiguration.

On the other hand, DPR allows the parts of the FPGA not being reconfigured to continue operating [37]. Also known as, active partial reconfiguration, DPR takes further what partial reconfiguration offers. While the time it takes to configure a part of the FPGA is very small, with frequent shutdown partial reconfigurations, there would be many pockets of small configuration times. For applications that require a high availability, for instance, datacentres, these pockets of configuration times could add up to constitute a significant downtime. DPR prevents such a scenario.

It is safe to state that DPR is the key enabling technology for a ROS as it forms the basis for runtime task management. The actual dynamic loading of tasks itself depends on DPR, and without runtime reconfiguration of the FPGA, it would be impossible for a ROS to offer a service like runtime error mitigation, which involves actively checking

the CMEM of the FPGA for bit flips and correcting any correctable bit flips all while tasks are actively executing on the FPGA.

A. The Process of DPR in Xilinx FPGAs

Figure 2.8 shows a diagrammatic representation of the Xilinx's PR scheme. The PR flow of Xilinx [37] involves the partitioning of the FPGA chip area into two main regions – the static region, containing resources not meant for PR; and the dynamic partially reconfigurable region. The reconfigurable region is then floor-planned into a number of *Reconfigurable Partitions* (RPs) to be shared by multiple *Reconfigurable Modules* (RMs). It should be noted that only RMs assigned to a particular RP can be configured in that RP. That is, intrinsically, there is no intrinsic support for relocating an RM from one RP to another. As will be seen in Section 3.2.4, this is an important requirement for reliability in terms of permanent on-chip resource damage circumvention.

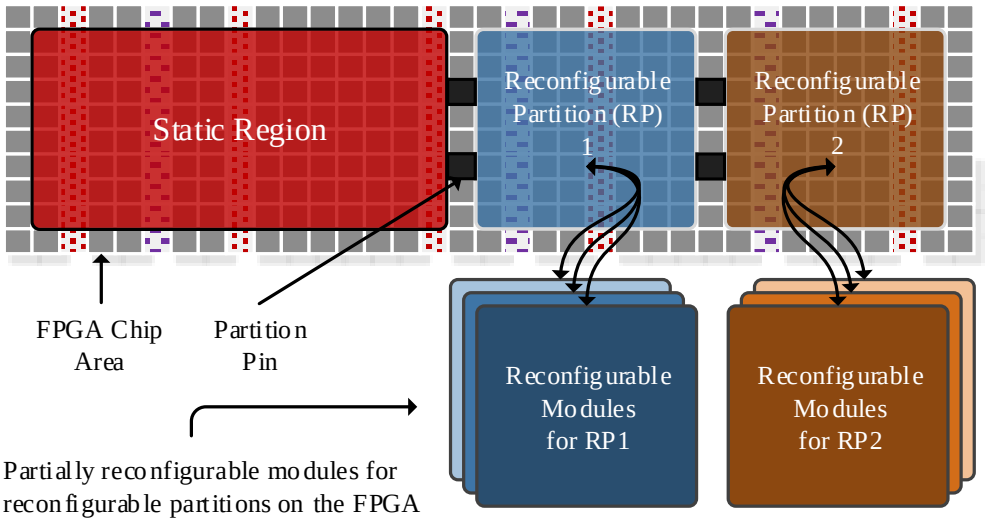


Figure 2.8: Diagrammatic representation of partial reconfiguration

For inter-communication between the circuits in the static region and the RMs in the RPs, and among the RMs (not recommended by Xilinx), during the PR flow, the build tool inserts LUT-based Partition Pins (PartPins) at every connection point between the static region and the RMs. The PartPins are LUT-based proxy logic that

serve as anchors in the RPs to provide a consistent interface between the static logic and the RMs. This interfacing approach is used by the vendor tool and it costs 1 LUT per signal connection in the early Vivado versions. This resource can be saved by using the PR link approach of [53], where blocker macros are used to force the tool to use a specific routing path for all static part and RM configurations, with the added advantage on improving relocatability of RMs. However, in the latest versions of Vivado PartPins incur no physical resources like LUTs but use the interconnect tile directly [37].

For each RM, a full bitstream and a partial bitstream (PB) are generated. The full bitstream is used for the first device-wide configuration of the FPGA mostly at power up, while the PB is used for (re)configuring the RM. More information on the PR flow of Xilinx can be found in [37].

It should be noted that the *reconfiguration frame*, which is the smallest selectable FPGA area for PR is two columns (CLB-CLB, CLB-DSP, or CLB-BRAM). This is because every two successive CLB columns (left CLB and right CLB pair) for instance, share an interconnect (INT) column, and partitions are not allowed to pass in-between them. Thus, the smallest selectable CLB area for an RP is two CLBs (right CLB and left CLB) as shown in Figure 2.9 [37]. The same applies to the boundaries between CLBs and BRAMs and between CLBs and DSPs.

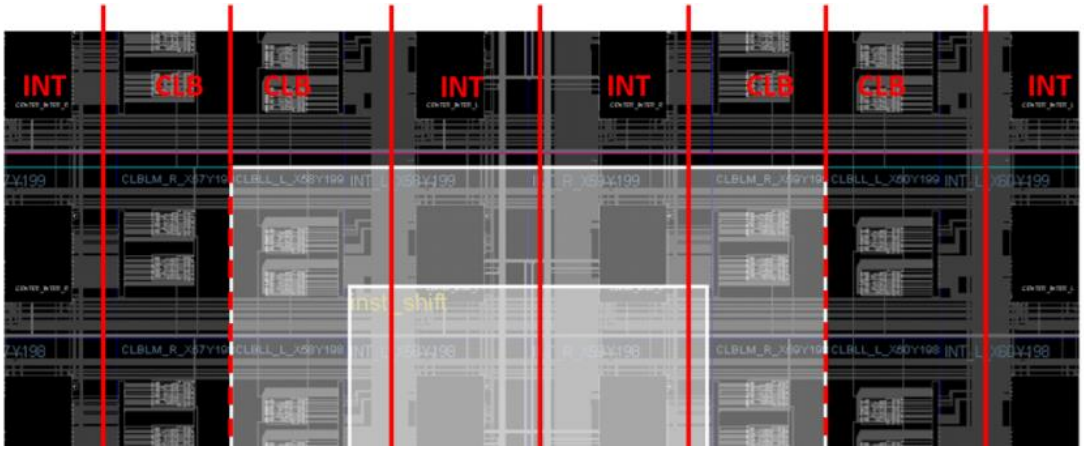


Figure 2.9: Permissible boundaries for a reconfiguration frame

The reconfiguration frame is important as it determines the granularity of the physical area selection for partially-reconfigured circuits. In contrast, the configuration

frame simply determines the granularity of configuration memory read and write. Table 2.11 presents the number of configuration frames required to (re)configure different reconfiguration frames on the 7 series FPGA. These numbers were obtained by studying the bitstreams generated for designs with different combinations of CLBs, DSPs, and BRAMs.

Table 2.11: Number of configuration frames for different 7 series FPGA resource pairs

Reconfiguration Frame	CLB-CLB	CLB-DSP	CLB-BRAM
Number of Configuration Frames	72	64	192

B. Module-Based Versus Difference-Based Partial Reconfiguration

Besides the partial reconfiguration flow that involves a whole RM being reconfigured in an RP, which can also be referred to as a *module-based* partial reconfiguration, Xilinx offers a form of partial reconfiguration which is *difference-based* [54]. The difference-based option is used for making small design changes to an RM. It is recommended for things like changing the equation of an LUT or modifying the memory content of a BRAM. If only a small change to an RM is required, instead of making a new RM that contains the change, and generating a new partial bitstream for the module, the small difference between the existing RM and the supposed new one is converted into a bitstream for reconfiguring only a small portion of the RM. The key advantage here is that the resulting difference-based partial bitstream can be much smaller than the RM's original partial bitstream. While the Xilinx BitGen toll can be used to generate difference-based partial bitstreams, with an understanding of the bitstream structure from reverse engineering analyses, it is possible to build such bitstreams using a custom tool.

2.3 Security and Integrity in FPGAs

Security in FPGAs has attracted a greater attention in recent years. This can be attributed to the more prominent deployment of FPGAs as computing platforms. FPGAs are susceptible to reverse engineering, cloning, and physical attacks among others [55]. While the methods and aims of attacks on FPGAs may vary, the result is

often malicious and as such, not desirable. As a result, the unifying theme in FPGA security is that of protecting the design and the device itself. As part of achieving this, FPGA vendors have relied on the complexity of bitstream formats as a deterrent to reverse engineering by attackers. The transformation of an FPGA-based design into a bitstream is a closely-guarded process, with no known successful reverse engineering attack. However, this is deemed as insufficient security for bitstreams.

2.3.1 Bitstream Security

Despite the fact that bitstream generation is a closely-guarded process, relying on the complexity of bitstreams as a deterrent to attackers is not considered prudent [43], and because there are other strains of attack like cloning, overbuilding, tempering, and spoofing, the best way of hiding design information is to use encrypted bitstreams. FPGA manufacturers have thus been driven to introduce encryption and decryption mechanisms into the FPGA configuration process [56].

Encryption is used to transform the readable *plaintext* bitstream into non-readable *cyphertext* bitstream. The encrypted bitstream is decrypted on-chip before delivering the data to the configuration memory. The encryption in the 7 series FPGAs is based on the 256-bit *Advanced Encryption Standard* (AES-256) implemented in the *Cypher Block Chaining* (CBC) mode, where a 256-bit key is used to encrypt and decrypt data blocks of 128 bits in size, with a CBC *Initial Vector* (IV) that is also 128 bits long. Figure 2.10 shows the AES encryption in the CBC mode.

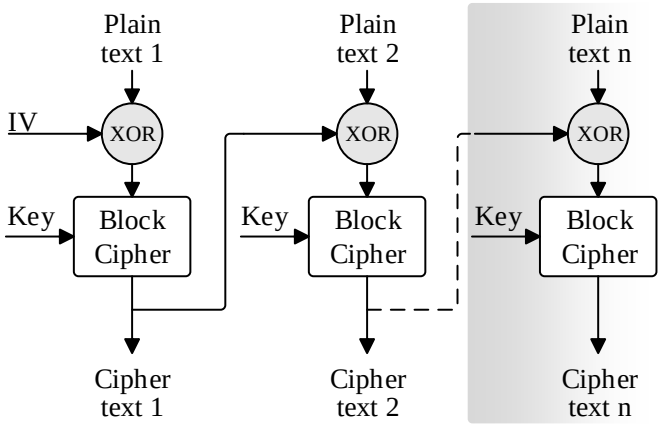


Figure 2.10: AES encryption in cypher block chaining mode

The CBC is a mode of block cypher encryption that hides recognizable patterns in the cyphertext by chaining the encryption of a plaintext block of data to the cyphertext of the previous block [57] (see Figure 2.10). It ensures that two identical plaintexts do not have similar cyphertexts. Since the encryption of each block requires the cyphertext of the previous block, a 128-bit IV is usually provided for the encryption and decryption of the first block. The process is reversed during the on-chip decryption, in which every decrypted block is *XORed* with the previous cyphertext block to obtain the plaintext block.

2.3.2 Bitstream Integrity

Despite the use of encryption, an attacker, even without a sufficient knowledge of the bitstream format can still manipulate portions of the encrypted bitstream with the aim of causing the FPGA to malfunction. This can happen if the validity of the bitstream is not ensured before configuration. It is obvious that encryption alone is insufficient. Therefore, to authenticate bitstreams and ensure the correct and intended operation of the device, hash algorithms have been applied in addition to encryption to ward off targeted malicious tampering [58].

A. Bitstream Authentication

Authentication ensures that the bitstream has not been deliberately or inadvertently modified before delivering it to the FPGA. For authentication in Xilinx FPGAs, the *Hash Message Authentication Code* (HMAC) is used. The Vivado applies a 256-bit HMAC key and the *Secure Hash Algorithm* (SHA) on the entire bitstream command and data to generate a 256-bit message digest or *Message Authentication Code* (MAC). The HMAC key and the MAC are both included as part of the unencrypted bitstream before encryption is applied. In the FPGA, as the bitstream is sent to the configuration logic after decryption, an on-chip HMAC-SHA-256 circuit uses the HMAC key to calculate the MAC on the configuration command and data and compares it with the MAC embedded in the bitstream. Any discrepancy results in the disabling of the configuration interface if the fallback option is not enabled [47]. As a result, the MAC helps determine the data integrity and authenticity of the bitstream. The HMAC-SHA-256 authentication is always enabled along with the AES-CBC-256 encryption. That

is, both encryption and authentication are applied together when the encryption option is selected. It is impossible to use encryption without authentication and vice versa.

B. Cyclic Redundancy Check (CRC)

For SRAM-based FPGAs, the configuration bitstreams are usually stored in an external non-volatile memory and transferred to the FPGA at power-up. The process of transferring the bitstream from this external storage can sometimes corrupt the bitstream. While the probability of this is low, the possibilities of bit flips occurring in the bitstream are particularly higher when bitstreams are delivered through error-prone mechanisms like radio transmission.

To ensure the correctness of the data, a 32-bit *Cyclic Redundancy Check* (CRC) is performed on the entire bitstream command and data during the bitstream generation. For the 7 series FPGA, the CRC-32 is calculated based on the CRC32C (Castagnoli) polynomial: $x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^{9} + x^{8} + x^{6} + 1$ [59] and is stored in the postamble of the bitstream. Xilinx uses this polynomial because it is more reliable at error detection than the default CRC-32-IEEE, evidenced by the fact that at a Hamming Distance of 4, it is able to cover a payload of 2147483615 bits. A CRC with a Hamming Distance of D allows all multi-bit errors below $D - 1$ bits to be detected [60].

While it may be similar to authentication, the CRC is not secure enough from the perspective of cryptography and is thus used primarily for error detection and control rather than data integrity confirmation [61]. When a bitstream is being uploaded to the configuration memory of the FPGA, an internal CRC-32 circuit calculates the CRC of the commands and data from the last CRC reset event. The calculated value is compared with the precomputed value stored in the bitstream. Any dissimilarity results in the configuration interface throwing a CRC error, which can be detected by monitoring the configuration interface's output, and reset by writing to the *Reset CRC* (RCRC) register.

C. Frame Error Correction Code (ECC)

In the 7 series FPGA, each CMEM frame is protected by a 13-bit *Error Correction Code* (ECC) stored in the 51st word of the frame while the entire CMEM is protected

by CRC. In reality, it is uncommon for an error to escape being detected by ECC and be found only by CRC [62]. This implies that in certain applications, the implementation of error mitigation could rely only on the frame ECC. The ECC bits are computed according to [63].

The FPGA comes with a FRAME_ECC primitive (depicted in Figure 2.11). If instantiated by the user in the design, this primitive enables the monitoring of the built-in ECC circuitry of the CMEM. Each time a readback is performed, the FRAME_ECC checks for errors by calculating a syndrome value from all the bits of the frame including the ECC bits in the 51st word. There is an error if SYNDROME[12:0] is non-zero. A single-bit error is indicated on the ECCERRORSINGLE port of the FRAME_ECC as a '1'. The index of the 32-bit frame word that contains the flipped bit is reported by SYNWORD[6:0] while SYNBIT[4:0] points to the bit position in the word. The CRCERROR port is used to signal a CRC error and is useful when the Readback CRC (see Section 2.3.2) is used.

The Frame ECC logic uses a *Single-Error Correction, Double-Error Detection* (SEC-DED) Hamming code, meaning that both singe-bit and multi-bit errors are detected but only single-bit errors can be corrected. The FRAME_ECC circuitry itself does not correct errors but indicates the location of a single flipped bit.

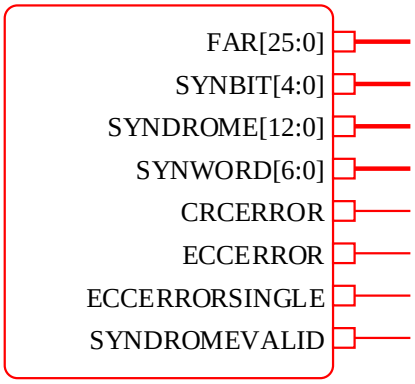


Figure 2.11: FRAME_ECC primitive's ports

2.3.3 Secure Bitstream Format

When encryption is used, the Xilinx Vivado tool generates the bitstream in a secure format (see Figure 2.12), different from the unencrypted version. The preamble is not

encrypted as it is used to set up the FPGA and the internal decryption circuit but the HMAC authentication key (HKEY) and the configuration data (which includes the configuration commands and frame data) are encrypted. There are eleven 32-bit *Use-Encryption* words in the preamble which consist of the commands to enable the AES decryptor; load the AES IV, and the *Decrypt Word Count* (DWC). The body section contains encrypted header commands, the frame address command and value, and the frame data. The encrypted body is appended with the HKEY. In the postamble, the footer commands and the HKEY, followed by the message digest (MAC) of the authentication process, are all encrypted.

The secure bitstream contains no DESYNC command packet (0x3000 8001 0000 000D) [47] as the DWC indicates when configuration should stop. The DWC is written with a command that has a register address of "11010" [47]. A more in-depth description of the secure bitstream format has been covered in [43]. This format can be confirmed by decrypting a Vivado-generated encrypted bitstream and studying its content.

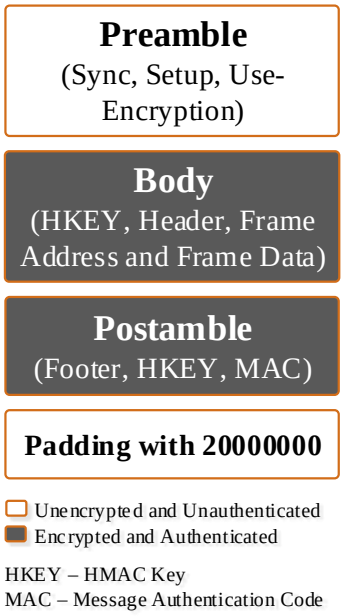


Figure 2.12: FPGA's secure bitstream format

2.3.4 Key Management

A key file (NKY file) is generated by Vivado when encryption is used. This file contains hexadecimal strings for the AES key, the IV, and the HMAC key. The system designer has the choice of supplying the keys and the IV or allowing them to be generated by implementation tool. To load the key file into the FPGA, the Vivado Hardware Manager can be used.

The AES key can be stored in the *Battery-Backed RAM* (BBRAM) or in the eFUSE. While the eFUSE requires no external battery, it is not as secure as the BBRAM. Once the key is programmed into the eFUSE, it cannot be cleared if there is an attack [64]. The IV is not stored on the chip but is used by the internal configuration circuit for the AES-CBC-based decryption of the bitstream being loaded. The IV is transparent to users and attackers since it is stored and transmitted unencrypted. The implication of this is that an attacker can modify the IV to cause malicious effects. However, because of the way the CBC scheme works, only the first block (128 bits) of the decrypted bitstream will be corrupted from being decrypted using an incorrect IV. Every subsequent cyphertext block uses is decrypted by chaining to the previous cyphertext (not plaintext) block, and as such will always be correctly recovered given that the AES key is correct.

2.4 Chapter Summary

Since their introduction by Xilinx in 1985, FPGAs have come a long way as computing devices. Starting out as a prototyping platform, they are now becoming a serious contender for mainstream computing. An FPGA can be considered as having three layers – *configuration*, *functional*, and *clock routing* layers. Using the Xilinx 7 series device as an example, this chapter has reviewed the different means of reconfiguring an FPGA. The key take away is that dynamic partial reconfiguration, which allows the FPGA to stay online and retain its operational state while a certain part of it is being reprogrammed, is an important technology that enables reconfigurable hardware to be deployed in ever-increasing number of applications.

The popularity of FPGAs means that the values of the IP cores that run on them has increased significantly. This has warranted the introduction of security measures including authentication and encryption to protect devices and the IP cores running on them. This however, has an impact on system design, with designers having to consider the storage of encryption keys. Moreover, the use of a special secure bitstream format will pose a challenge to design techniques that require the bitstream to be in plain formats. This is the motivation for the work in Chapter 5, where a unique secure bitstream and supporting configuration controller and formatting software are advanced.

Chapter 3

Reconfigurable Computing

3

In the 1940s, the need for flexibility in computation led to the introduction of the computer processor or CPU [65][66] for stored-programme as against fixed-programme computing. However, no sooner had the processor been introduced, than it was realized that it was limited in performance. Early CPUs ran at relatively slow frequencies with very limited performance compared to what is obtainable today. However, as the demand for computing increased, it was imperative to increase the speed of processors. By 1971, the first commercial microprocessor, the Intel 4004, which integrated a CPU comprising of 2300 transistors into a single chip, had appeared [67]. The advent of microprocessors drastically improved speed and performance. For years, the integrating density, which fuelled an increase in the performance of microprocessors, kept increasing, doubling every two years, as predicted by Moore's law, but a limit in practicable processor speed was soon reached, as the enormous heat generated by microprocessors could not be extracted fast enough; a phenomenon known as the *Power Wall*. Keeping the voltage down is a way of reducing power dissipation but this is limited by the unavoidable transistor leakages and process variations in the fabrication techniques [8]. This has prompted the move from uniprocessors to multi-core processors and multiprocessors. While this masks the original problem of the processor's software-based computation which arises from the sequential execution of processes and the von Neumann bottleneck [5], this is also destined to ultimately slam into the *Power Wall*.

In contrast, a hardware-based computation has the capacity to execute multiple tasks in parallel and thus, provide a high computational performance at low power. For instance, in comparison to processors, FPGAs have been shown to be up to 540 times faster [68] and up to 70% more power efficient [69] for certain applications. The trend is thus, towards the use of specialized hardware to execute compute-intensive and time-critical tasks while the host processor focusses on software tasks. This can be seen in the latest Xilinx and Altera heterogeneous System-on-Chip (SoC) devices, combining

a high-performance ARM processor and an FPGA fabric into a single chip [70][71]. While specialized hardware based on *Application-Specific Integrated Circuits* (ASICs) offers the advantage of speed, it lacks the flexibility of software. Reconfigurable hardware like the FPGA, on the other hand, offers programmability to logic circuits, combining the *flexibility of software* with the sheer *high performance of ASIC-like hardware*. The subject of reconfigurable computing is that of exploiting this powerful combination to meet high computing demands, reduce cost, and achieve lower power consumption.

Years before FPGAs were conceived, the earliest reconfigurable computing architecture was proposed by Gerald Estrin in 1960. In his work on the fixed-plus-variable structure computer [72][73], he proposed a reconfigurable computer architecture that would "permit computations which are beyond the capabilities of [the then] present systems by providing an inventory of high speed substructures and rules for interconnecting them such that the entire system may be temporarily distorted into a problem oriented special purpose computer" [72]. However, it was not until 1984 that the first real reconfigurable computing hardware fabric appeared when Xilinx introduced the first FPGA and since then FPGAs have progressed through various stages of development [10]. From that humble beginning of the XC2064 FPGA which contained about 64 logic-only cells (less than 1000 gates), current FPGAs now boast of millions of gates making up heterogeneous resources that include not only logic cells but also memory elements, processing elements, and other functional blocks.

There are other reconfigurable hardware like *Programmable Array Logics* (PALs) and *Complex Programmable Logic Devices* (CPLDs) but FPGAs are the most widely used because of their higher densities, allowing for highly complex designs. In 2017, the global FPGA market accounted for $63.05 billion and is forecasted to reach up to $117.97 billion by 2026 growing at a compound annual growth rate of 7.2% [74]. The driving force of this market is the adoption of FPGAs for various applications ranging from hand-held devices to datacentres and aerospace.

Several concepts and methods have been advanced in a bid to exploit the potentials of reconfigurable hardware, especially FPGAs. The research efforts expended have led to a number of promising architectural solutions in the form of

ROSes that aim to present the FPGA fabric as an extension of conventional processors. However, in the various architectures, there is a uniformity of purpose, which is the abstraction of the intricacies of the reconfigurable hardware fabric by providing essential RC services. In the following sections, the existing RC architecture and service implementations will be brought to light and considered in the light of the objectives of this research. Furthermore, several works have implemented key RC services without focussing on full ROS implementations. These will also be reviewed.

3.1 An Overview of Reconfigurable Operating Systems

The introduction of DPR in FPGAs allows parts of a device to be reprogrammed while other parts are operating (see Section 2.2.2). Using this capability, hardware circuits can time-share resources and be swapped in and out of the FPGA as and at when required in runtime; and following the software nomenclature, these circuits can be named as "hardware tasks". A strong requirement for the categorization of a hardware circuit as hardware task is the sharing of resources and intercommunication with other circuits. Where this is not the case, a hardware circuit in a reconfigurable system would be best described as a *hardware accelerator* [19]. New terminologies like *morphware* and *configware* have also been coined to describe a reconfigurable hardware device and the application mapped onto it [75].

A *Reconfigurable Operating System* (ROS) attempts to give to hardware a *software look* by abstracting the low-level intricacies of hardware from application developers, and thereby easing the development of applications for execution on FPGAs. Traditional operating systems for processors manage every aspect of the operations of processors, from process execution to memory management, and I/O access. In a similar vein, a ROS aims to manage the on-chip resources in an FPGA, but mostly on behalf of software processes running in a host CPU for an improved system-wide performance. In general, a ROS is an extension of a conventional software operating system.

Typically, a ROS is expected to provide services ranging from task scheduling, task placement, task configuration, inter-task communication, partitioning, memory

management, IO, and security [22]–[24]. Arguably, the most essential of these are task placement, task configuration and inter-task communication. Depending on the chip area partitioning approach used by a ROS, three reconfigurable area styles are possible in ROSes [24]. In the *island* style, a number of RPs can be marked out on the chip for hosting one task each without the possibility of overlapping; that is, an RM or task cannot use multiple partitions. Most existing ROSes use island-style partitions. On the other hand, in the *slot* style of reconfigurable area, the chip is divided into contiguous areas that allow one-dimensional placement of tasks. In a closely related style, the *grid* allows a two-dimensional placement. In both the slot and the grid styles, an RM can occupy multiple RPs.

The main concerns in this thesis relate to task configuration and communication. As such, how these and closely-related features are implemented in the sate-of-the-art ROSes will now be reviewed. OS4RS [76] is an operating system primarily for software-to-hardware context switching and vice-versa. Each task is designed to have software and hardware versions with the same functionality. The reconfigurable fabric is divided into fixed islands called *tiles*, which are of the same shape and size. The tiles are interconnected by a network-on-chip (NoC), with each tile designed to run a single task at any given time. For task loading, partial reconfiguration is carried out through the ICAP.

In [77] and [78], the authors present HybridThreads (HThreads), which is a real-time OS kernel that allows hardware and software multithreading. Inter-task communication is provided over a system bus using a memory-mapped register interface. It is difficult to classify HThreads as a complete ROS since it does not support DPR and as a result, there is no resource sharing among the threads (tasks) [79]. The hardware tasks are never reconfigured. Instead, they are static tasks that reside in static slots attached to the system bus.

BORPH [80], developed at the University of California at Berkley, is another Linux extension. In BORPH, a whole FPGA is used as the reconfigurable island, with four FPGAs interconnected to serve as user FPGAs while one FPGA serves as the control FPGA. However, using a whole FPGA as a reconfigurable slot can easily lead to inefficient resource utilization and high reconfiguration time overheads, especially

since the FPGAs are not partially reconfigured. For communication, BORPH uses a SelectMap bus connection between the central control FPGA and the four user FPGAs while inter-FPGA communication is *Point-to-Point* (P2P) via a ring network.

ReconOS [81] [82], CAP-OS [83], FUSE [84], SPREAD [85], and RTSM [86] are further ROSes that all generically use island-style partitions and provide communication by means of P2P connections, buses, or NoCs. However, in [87], a grid partitioning is provided targeting the ReconOS. In addition, all these ROSes support DPR. However, while FUSE claims support for DPR, in the proof of concept implementation, the hardware tasks are fixed and not dynamically reconfigured.

Rainbow [88] is an extension of a real-time OS kernel for hardware multitasking. While the real-time OS provides software multitasking functionality, the Rainbow extension adds a hardware multitasking capability. Fixed slots in grid style are provided for placing hardware tasks and communication is provided by a P2P scheme. It provides an ICAP-based configuration controller for hardware task allocation and pre-emption.

All of the afore-mentioned ROSes are in line with FPGA vendor design tools, which force the use of fixed islands and slotted partitions to preserve the routes between the static and reconfigurable regions (see Section 2.2.2). However, fixed partitions lead to inefficient use of FPGA resources (e.g., small circuits must be mapped to enlarged reconfigurable slots) and limits the chances of finding a damage-free alternate location for hardware tasks (e.g., a single damaged resource makes an entire slot unusable). In addition, as seen in the afore-mentioned ROSes, reconfigurable slots are typically interconnected by means of a static communication infrastructure (e.g., bus or NoC), which does not only consume a significant amount of FPGA resources [89], but also fills the chip with lots of static routes that limit the allocatability of hardware tasks.

As opposed to these approaches, R3TOS [79], advanced to be a reliable ROS, uses a slotless reconfiguration mode in which hardware tasks can be arbitrarily placed on the FPGA. R3TOS does not rely on any fixed communication infrastructure like P2P, buses, or NoC. Instead, it provides inter-task communications and synchronization through the configuration layer by using the ICAP to copy data between buffer

memories attached to the inputs and outputs of tasks [90]. While R3TOS can be classified as grid-based in terms of reconfigurable area style, it is in fact different by definition as it does not actually mark out slots on the chip area. R3TOS keeps the FPGA reconfigurable area empty, that is, free of any partition boundaries (i.e., PartPins) and static routes. However, R3TOS relies heavily on DPR for task loading, deallocation, and inter-task communication & synchronization.

Table 3.1 summarizes the relevant features of all the ROSes just discussed. The common feature in all of these ROSes, except Rainbow and R3TOS, is the use of islands for task placement. This is a consequence of the nature of the infrastructure used for inter-task communication. Except R3TOS, the other ROSes provide P2P, bus, or NoC-based communication, which uses the resources in the functional layer for implementing communication. The choices of partitioning and communication are influenced by the limitations imposed by the FPGA design tool flows and the increasing heterogeneity of modern FPGAs [88]. To address this, R3TOS deploys the configuration layer for communication by using the ICAP to relocate data between tasks. Hence, it is able to advance a slotless architecture that favours relocatability.

Table 3.1: Architecture of existing reconfigurable operating systems

ROS	**Reconfiguration Style**	**Architecture**	
		Reconfigurable Area Style	***Inter-Task Communication***
OS4RS	DPR	Islands	NoC
HThreads	None	Static islands	Bus
BORPH	Full Reconfiguration	FPGA-based islands	P2P
ReconOS	DPR	Islands/grid	Bus
CAP-OS	DPR	Islands	NoC
FUSE	DPR	Islands	Bus
SPREAD	DPR	Islands	Bus and P2P
RTSM	DPR	Islands	Bus
Rainbow	DPR	Grid	P2P
R3TOS	DPR	Grid	ICAP-Based

While the provision of the key RC services has been generally provided by the afore-mentioned ROSes, implementing these services in a ROS expected to be *reliable* brings a whole new set of technical challenges. For instance, apart from R3TOS, there is no other ROS that has laid claim to being reliable in the context of runtime error mitigation, especially for permanent damage circumvention. By far, R3TOS can be considered as the only ROS that has put a strong focus on reliability. Mostly, the inability to advance a ROS with high reliability can be attributed to the limitations imposed by the communication mechanisms employed by these ROSes. They are all based on statically determined inter-task communication infrastructures that are not amenable to runtime task relocation for permanent damage circumvention.

While R3TOS attempted to solve the communication bottleneck by using the configuration infrastructure of the FPGA for inter-task data transfer [90], other critical services relying on configuration are impaired, as the configuration interface is a single resource that needs to be shared by these critical system services [83]. As a result, one of the major issues addressed in this work is that of *inter-task communication for reliable reconfigurable computing* (see Chapter 6).

3.2 Reliability Concerns in Reconfigurable Computing

Since FPGAs now find use in high-value applications, the requirement for reliability is all the more important. Radiations are able to cause *errors* like temporary upsets or permanent latch-ups in the underlying physical structures (e.g., CMOS transistors) of the FPGA, propagating *faults* to the CLBs, BRAMs, and DSPs that make up the digital designs resident on the FPGA, and potentially leading to system *failures*. These errors can as well appear in the CMEM cells and other device-functionality primitives. Resilience to these errors is an integral factor in the design of reliable reconfigurable hardware systems. The following sections will provide further insights into the nature and causes of these errors and the techniques deployed by system designers to combat the menace.

3.2.1 Soft and Hard Errors

The underlying fabric of SRAM FPGAs is fabricated using the CMOS technology. Most ICs today, including reconfigurable devices use the CMOS technology because of its high density, high speed, and low power consumption [91][92]. The CMEM in SRAM-based FPGAs are composed of CMOS SRAM cells, with a typical cell made up of six transistors. The interconnect switches, CLBs, BRAMs, DSPs, and other primitives, including control registers on the FPGA are made from CMOS transistors. The correct functionality of the device and the digital design built from these resources depends on the ON or OFF logic states of these transistors. As such, an unintended bit flip, which turns an ON transistor OFF or vice-versa is able to cripple the functionality of a digital design mapped on the FPGA. Such a bit flip can be temporary (*soft*) and would disappear when the device is power-cycled, or permanent (*hard*), in which case there is a permanent damage to the transistor and any digital design built on it would be permanently affected.

Compared to hard errors, which render affected FPGA resources permanently unusable, soft errors can generally be corrected without taking the device offline. It is important to correct soft errors as soon as they appear because if left to accumulate, they may become uncorrectable, and a full device configuration might be unavoidable, increasing the downtime of a device or system. The main cause of errors is radiation in space and nuclear environments. The description of the underlying physical processes and interactions that trigger errors have been covered in [93].

A. Soft Errors

A *soft error* occurs in an FPGA when one or more bits of the chip's transistors flip due mainly to charge disturbance from ionizing radiations [94], especially for devices deployed in space and nuclear applications. A single ionizing energetic particle can transfer an electrical charge to a device to cause a *Single-Event Effect* (SEE), which can be soft or hard. On the other hand *Total Ionizing Dose* (TID) effects are a result of accumulated ionizing energy deposited by photons or particles getting trapped in the insulating materials of transistors [95], and inducing a degradation of the device with time.

There are different types of SEEs, depending on the kind of effect created. When a soft error causes a single bit flip in a storage element like memory cell, latch, or flip-flop, it can be referred to as a *Single-Event Upset* (SEU) [96] and is the most common of all SEEs. This is in contrast to the less likely *Multiple-Bit Upset* (MBU) where more than one bits are affected by a high-energy radiation.

A *Single-Event Functional Interrupt* (SEFI) is a result of an SEE that affects a critical system control register or an internal circuit needed to operate a device and in the process causes the entire device to malfunction [97]. Six types of SEFIs have been identified for the Virtex-4 FPGA, but are as well applicable to other FPGA families. These are: Power-On-Reset (POR) SEFI, SelectMAP (SMAP) SEFI, Frame Address Register (FAR) SEFI, Global Signal SEFI, Readback SEFI, and Scrub SEFI. The detail descriptions of these SEFIs can be found in [98]. SEFIs are critical since a user design cannot correct them by scrubbing in most cases. However, they rarely occur, with approximately one SEFI occurring in 65 years for galactic cosmic rays in deep space [99].

When a logic gate is hit by an SEE, the corresponding event is referred to as a *Single-Event Transient* (SET) as this leads to an error only when an erroneous bit state propagates through one or more combinatorial path and eventually gets registered in a memory element [96]. All the afore-mentioned soft error modes are non-destructive as they do not potentially lead to permanent damages. A *Single-Event Latch-up* (SEL) is a potentially destructive SEE "where a low-resistance path develops between power supply and ground but remains after the triggering event is removed" creating high-current conditions that can potentially destroy a device [100].

In the functional layer of an FPGA, BRAMs are more highly prone to radiation effects while flip-flops are less susceptible because they constitute the smallest portion of the internal state of the FPGA. On the other hand, in the configuration layer, because of the sheer number of bits the CMEM contains, it is comparatively more affected by radiation effects, with bits mapping logic resources less vulnerable compared to those of the routing resources [101][102].

B. Hard Errors

Hard errors are permanent damages or defects that appear in the underlying silicon fabric of devices. These errors are permanent and cannot be remedied by any method used for mitigating soft errors. Since these errors cannot be corrected, if they happen to fall within the region occupied by an active task, they would most likely lead to computational errors or eventual system failures depending on the essentiality of the bits affected.

The continuous process scaling leading to the miniaturization of transistors has given rise to FPGAs with ever higher densities than before. The repercussion however, is that the on-chip resources and interconnects have become more prone to degradation and ageing-related permanent damages or hard errors. For instance, [103] reports that resources on Xilinx 65-nm-node FPGAs can experience first-time failures within three to five years of the device going into operation. While hard errors can be caused by manufacturing defects, the most important from the user's perspective are those caused by ageing-related factors, including: *Time-Dependent Dielectric Breakdown* (TDDB), *Electromigration*, *Negative Bias Thermal Instability* (NBTI), *Hot-Carrier Effects* (HCEs), and Stress Migration. The description of the physical processes powering these mechanisms have been covered in [103]. Another factor that can influence ageing and eventually cause damages is thermal cycling (extreme variation of temperatures) in the chip substrate [104].

Ageing-related errors can occur due to the natural unavoidable wear and tear or degradation of the silicon as time passes. However, radiations can cause SELs, which can trigger unintentional excessive currents and eventually leading to permanent damages [100][105]. Furthermore, radiation can cause permanent errors in the form of *Single Event Gate Rupture* (SEGR), or *Single Event Burnout* (SEB) [106]. In addition, permanent failure modes earlier discussed can be exacerbated by extremes of temperatures [107].

3.2.2 Soft Error Mitigation (SEM) in FPGAs

SEM includes both the detection and correction of soft errors. The early detection of errors is important in reliable systems, allowing for mitigation techniques to be applied.

If errors are allowed to accumulate, the system might eventually need to be power-cycled to clear the errors, contributing to system downtime. Error detection techniques that permit a runtime diagnosis and correction are thus highly favoured in reliable dynamically-reconfigurable computing devices. There are several methods for mitigating errors, which include radiation hardening and redundancy. An in-depth survey of mitigations techniques can be found in [108].

A. Radiation Hardening

Radiation hardening is any process or technique applied to make a device resistant to ionizing radiation effects. There are two general categories – *Radiation Hardening by Process* (RHBP) and *Radiation Hardening by Design* (RHBD) [109]. RHBP involves making changes at the fabrication level in silicon. It involves modifying fabrication processes. This is done in order to make ICs withstand radiation doses in the order of 100 krad to more than 1 Mrad for space-grade use [31], with the majority of space applications requiring a tolerance of 300 krad (Si) for TIDs and 37 MeV-cm^2/mg for SEUs [110]. However, the high cost of these ICs and the long lead times are a disadvantage [111]. Moreover, the fabrication process deployed for RHBP is a number of years (two generations or more) behind state-of-the-art [112], with the implication that RHBP devices have inferior density and performance.

An alternative to RHBP is RHBD, which is the use of design techniques based on standard CMOS technologies to harden ICs without making fabrication process changes. RHBD approaches range from gate-level, techniques like selectively hardening the critical gates (those affecting a circuit's functionality) of a circuit [113] and circuit-level [114], to system-level methods [115]. Since gate-level mitigations are applied during fabrication, they are not applicable to COTS devices which are already in the market. As such, error mitigation approaches for COTS FPGAs are often at the circuit (e.g., flip-flops and LUTs), module, and system levels.

On the other hand, as most COTS devices can tolerate radiation doses in the range of 5 krad to 20 krad or more, it is possible to use them in radiation environments when properly characterized with "*Careful COTS*" design steps [31]. This radiation tolerance, which is observed in parts not manufactured with space radiation in mind can be raised by deploying system-level reliability approaches similar to the ones used in RHBD. In

particular, the flexibility of reconfigurable hardware can be exploited to raise the tolerance levels of COTS FPGAs in a bid to make them usable as space-grade devices within a characterized radiation dose and SEE tolerance range.

B. Triple-Modular Redundancy

A classical method of mitigating errors is *Triple-Modular Redundancy* (TMR) with majority voting [116][117], where three instances of a circuit are configured. A voting mechanism is then used to compare the outputs of the three circuits. The correct output is the same as that of at least two of the three instances. The likelihood of having two instances being struck by error in the same manner and thus, producing the same erroneous outputs, is highly unlikely. However, where system reliability requirements are higher, a designer would do well to revert to another error detection mechanism as backup. To avoid a single point of failure, where the majority voter itself fails, the voter can be triplicated [118].

TMR can best be described as an error detection and masking technique, its main advantage being that it ensures a continuous correct functionality in the presence of errors. Essentially, it masks the underlying error and prevents it from propagating faults into the rest of the system. However, if the errors are not cleared (corrected) as soon as possible, the TMR scheme could eventually be defeated as errors accumulate. Two modules could become faulty at a time, and all the three modules could produce three different outputs, defeating the voting mechanism.

TMR is applicable to both soft and hard error detection. However, as good as TMR is, it has its failings. The apparent downside of a traditional TMR implementation is that it incurs a large overhead, at least 200% more resource and area utilization arising from the two redundant modules. To mitigate this, several variants of TMR have been implemented to cut down on the resource overhead. For instance, in [119], a selective TMR (STMR) mechanism is introduced, where SEU-sensitive sub-circuits of a main design are identified and triplicated. The STMR technique achieved an area overhead of 60-70% of that of the traditional TMR approach for the example circuits used.

For radiation hardening, TMR can be applied at the gate, circuit, module, device, or system level [118][120]. However, as TMR does not correct bit flips and errors

should not be allowed to accumulate, other mechanisms must be deployed for the correction of errors as they emerge. A popular method for soft error correction is *scrubbing*, which involves overwriting the CMEM through PR to correct bit flips [121]. Scrubbing is useful for mitigating errors in the CMEM and can be external if the scrubber is outside the FPGA and internal, if otherwise. In terms of the approach to updating the CMEM, two scrubbing methods have been identified – blind scrubbing and readback scrubbing. In fact, TMR can be combined with scrubbing to obtain a substantially reduced failure rate [122].

C. Blind Scrubbing

To prevent error accumulation, a technique called scrubbing is often used. Scrubbing involves a continuous update of the CMEM as often as possible with a *golden* bitstream stored in a rad-hard memory outside the FPGA. It is recommended to be carried out at least ten times faster than the worst-case SEU rate of the target FPGA [99]. Since the CMEM is refreshed often, any soft error in the CMEM is eventually corrected. However, care must be taken not to overwrite user data in the flip-flops, LUT-RAMs, *Shift Register LUTs* (SRLs), and BRAMs as the contents of these could have changed since configuration. This method of scrubbing has been referred to as *Blind Scrubbing* since the CMEM is updated without first reading the CMEM to check whether or not there has been an upset.

Though fast and simple, blind scrubbing can cause a *Scrub SEFI* and risk damage to the device as observed in [123], where an SEU in the FAR register or a SET on the clock line feeding the configuration data caused frame data to be written to a wrong location and triggered contention which led to high currents on the core supply. Since a SEFI cannot be scrubbed, a solution to Scrub SEFI is to perform the scrubbing frame-by-frame rather than full-device, with the implication that when an upset is detected, only a single frame would be written. This is instead of allowing the FAR to auto-increase from an initial starting value, which is the usual practice with blind scrubbing. This solution is proposed in [123] and evaluated in [124].

Another shortcoming of blind scrubbing is that if the internal scrubbing approach is used and the ICAP is relied upon for writing the golden bitstream, as is often the case,

this reduces the availability of the ICAP for reconfiguration. Moreover, the need for a rad-hard external memory to keep the golden bitstream creates an added cost.

D. Readback Scrubbing

With the advent of DPR in FPGAs, an alternative approach to blind scrubbing has emerged – *Readback Scrubbing*. In readback scrubbing, the CMEM frames are first read back and compared with a golden bitstream to check for errors. If there is an error, the golden frame data is written to the CMEM to correct the error. While this method still requires an external memory, the advantage is that there is no need for continuous writing. However, the ICAP is still continuously occupied for readback operations.

Along the lines of readback scrubbing, the Xilinx FPGAs, starting from the Virtex-4, have shipped with a FRAME_ECC primitive that when activated, computes a syndrome value to determine errors in frames during readback. This can be used to eliminate the need for an external storage, although only single errors can be corrected. The Xilinx SEM IP [62] and the work in [125] circumvent this single-error correction limitation by allowing frame data to be loaded from an external memory to correct multi-bit errors especially since these can constitute a significant portion of errors detected, as observed in [126]. Further details on the FRAME_ECC primitive have been presented in Section 2.3.1.

In the 7 series FPGA, there is an internal *Readback CRC* circuitry that can automatically perform readback in the background of user design to detect and correct SEUs in the CMEM [47]. It is based on a CRC check of the CMEM. The Xilinx SEM IP [62] and the work in [125] both rely on this internal circuitry. While Readback CRC eliminates the need for a custom readback circuitry, it, however, cannot be controlled to scan only a specific number of frames in the CMEM. As at the time of writing, Xilinx documentations have not proven this to be otherwise. This means the entire chip has to be scanned in each SEM pass. Earlier, we noted that less than 1% of the ICAP's bandwidth should be used for configuration if SEM coverage of the entire chip is desired. To put this in perspective, we note that the smallest task on the FPGA will occupy a CLB-DSP pair and this requires a total of 64 frames for configuration (see Table 2.11 in Section 2.2.2). These frames can be written in 66.24 µs (computed from Table 4.15 of Section 4.5). Taking the XC7Z100 chip as an example, the maximum

full device scan time of the chip is 34.3 ms [62]. If the 66.24 µs is taken as 1% then SEM will take 6.56 ms. To ensure an effective SEM coverage, 6.56 ms has to be the minimum duration between successive task reconfigurations. However, practical hard real-time systems can have time bounds on the order of several microseconds to a few milliseconds [127], implying that at > 99% ICAP occupation for SEM, the system will be operating outside the bounds of hard real-time processing.

E. Diagnosis of User Memories

After the initial configuration of a circuit, the user data in the flip-flops and BRAMs can change as the circuit operates. These changes are not updated in the corresponding bits in the CMEM. As such, if a flip-flop or BRAM bit is hit by a SEU, it is impossible to detect the flipped bit by checking the CMEM. For mitigating errors in the BRAM, the BRAM in the 7 series FPGA comes with a built-in ECC that supports SEC-DED, similar to the FRAME_ECC for CMEM. As ECC logic does not actually correct the error but only presents a correct memory output, a form of memory scrubbing is also recommended for ECC-protected BRAMs [102].

LUTs and other logic-related resources are susceptible to SETs. Since they have no memory, the transient event does not constitute an error until registered in a memory element like flip-flop or latch. As such, it is difficult to distinguish between a bit flip that is a direct result of an SEU or one that is indirectly triggered by a SET. To prevent transient events from inducing SEUs, SET filters can be used [128]. For flip-flop error mitigation while TMR can be used to mask error, the isolation of the erroneous bit often requires a self-checking mechanism like *Built-In Self-Test* (BIST) circuitry.

While TMR can be used to mask an SEU in a flip-flop and feedback from the voter used to correct it [108], this is applicable when TMR is applied to individual flip-flop. When an entire module is triplicated, it becomes more challenging to fix the error. The entire module can be reconfigured but instead, a BIST circuit can be configured in the region under test to identify the offending flip-flop and reconfigure only the associated configuration frame. The use of BISTs prevents the need for an external diagnosis service. BIST circuits have been proposed for diagnosing LUTs, flip-flops, and BRAMs [129], with some concerned more with reducing the storage requirement and

configuration overhead for BIST circuit bitstreams by using a BIST-cloning technique [130] while others consider the reduction in test time in addition [131].

3.2.3 Hard Error Mitigation (HEM)

The mitigation of hard errors or more appropriately, permanent damages, that appear on the FPGA chip area ranges from delaying the occurrence of the errors, to circumventing them when they eventually surface. Mitigation techniques include: radiation hardening (e.g., using TMR for error masking) and BIST for error isolation. These have already been discussed in Section 3.2.2. However, in addition to diagnosing flip-flops, BIST can also be used to detect permanently damaged logic elements (e.g., LUTs) [130] and routing resources [132]. In addition, when a BIST detects a hard error, the error cannot be corrected by reconfiguring the affected frame. Instead, the affected resource is completely avoided. Other mitigation approaches, relevant especially to HEM, include *Wear Levelling* (WL); online rerouting; and precompiled circuits for alternate positions.

A. Wear Levelling (WL)

To mitigate ageing-related hard errors, wear-levelling (uniform-ageing) strategies can be applied. WL is any strategy deployed to pre-empt hard errors before they happen. Eventually, every silicon chip will wear out; however, prolonging the lifespan of the chip is crucial and this is the aim of WL. In [103], the authors proposed different WL methods to achieve uniform ageing of FPGA chips with respect to different causative factors. Load balancing was used to tackle the impact of HCEs by periodically changing the physical location of circuits to even out the usage of resources over time. *Selective Alternate Routing Technique* (SART) was proposed for increasing the time to failure due to electromigration in interconnects through dynamically switching between an ageing route and an alternate unused route between logic resources (see Figure 3.1, where an alternate route is used for connecting between the two logic blocks). Using the SART technique, a percentage increase in chip age of up to 40% was observed in an example benchmark. Another solution was configuration bit flipping for alleviating the impact of NBTI. All these mitigation techniques were simulated and reported to

extend the operating lifetime of an FPGA chip, even up to a 100% in one of the benchmark designs.

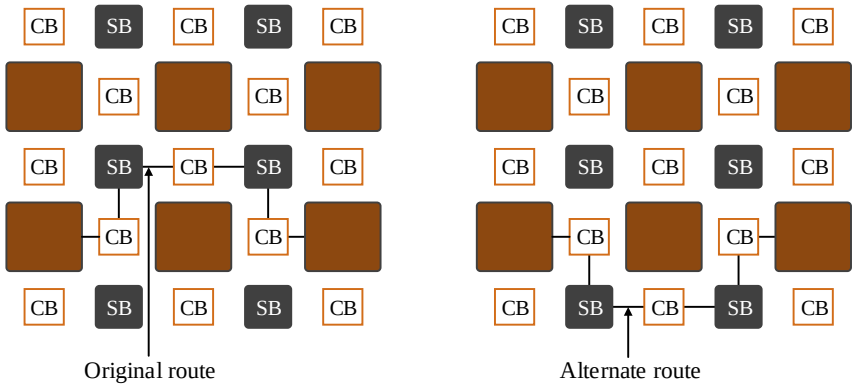


Figure 3.1: Alternate routing as a wear-levelling strategy

The authors in [133] argued that the findings of [103] were based on simulations and were therefore difficult to verify. They went ahead to propose three WL strategies, namely: *alternative logic mapping*, *spare resources*, and *alternative placement* to mitigate the effects of NBTI. Since static logic levels are the root cause of NBTI, these methods respectively involve disrupting constant logic levels by inverting the sense of inputs and outputs in the design to alter the stress pattern, relocating active circuits to unused resources, and exchanging the positions of circuits to ensure that logic resources are stressed under multiple similar conditions. Hardware evaluations of these three WL strategies using accelerated-life tests demonstrated that at least a 20% reduction in degradation can be obtained.

A solution to thermal cycling is proposed in [104] which involves cyclically relocating circuits with high power dissipation to regions where tasks with low power emissions are allocated and vice-versa. As the temperature of a region on the chip directly obtains from the power consumption of the circuit occupying that region, this means unoccupied and less-used regions will have lower temperatures. To even out the temperature of the chip and prevent hot spots or a steep temperature gradient, circuits are moved about the device intermittently at a cycle period determined by the switching frequencies and the area occupation of the circuits.

Though the prime advantage of WL for FPGAs is that there is no need to actively check the chip for errors and address them in run-time, WL cannot be solely relied upon since the eventual occurrence of age-related hard errors is inevitable and unpredictable. As such, WL should be deployed as a *passive* mitigation technique while *active* strategies should be used for periodic hard error detection and mitigation.

B. Precompiled Circuits

Precompiled circuit configurations can be used to target alternate locations in the event of damages. This involves anticipating that damages will occur and mapping circuits to multiple pre-determined regions. In a basic form of this, a battery of bitstreams corresponding to different regions for each circuit are generated. This idea is used in [134], where multiple test bitstreams are used to avoid defects in a programmable device. The obvious downside to this is the increased bitstream storage requirement.

To reduce bitstream storage size, in the column-based precompiled configuration technique of [135], alternative RMs that avoid specific areas of the original RM are created by shifting the original RM one column at a time to create similarities among generated bitstreams. Differential coding and data compression are then used to reduce the ensuing bitstream size. The implementation however, does not cater for the possible resource heterogeneity of FPGAs as only CLBs are considered.

C. Online Rerouting

Online rerouting is another method used for circumventing permanent damages. In [136], the authors propose an incremental routing strategy for ripping up and rerouting interconnections between logic blocks. This can be used to provide communication support for circuits targeting alternate locations. The major downside of online rerouting is the prohibitively high time it incurs. Even with the efficient incremental rerouting implementation of [137], with reconfiguration time per fault as high as 73 seconds.

Because of the high computational cost needed, routing algorithms are often run offline. As such, in the rerouting strategy of [137], an external software is used to determine the new route before the actual on-chip rerouting. On the other hand, self-

routing approaches have been proposed. These do not require auxiliary external software as demonstrated in [138] where faulty cells are automatically replaced with spare ones and rerouted. However, self-routing can be at the expense of increased resource usage due to the self-routing core and some approximations or compromises might have to be made as exemplified by [139].

3.2.4 Partial Bitstream Relocation

Most of the proposed solutions to HEM require first, the ability to reconfigure the FPGA dynamically, in which DPR comes to the rescue; and second, the ability to move a configured circuit from one place to the other on the chip in runtime [140], often referred to as *Partial Bitstream Relocation* (PBR). In the context of RC, PBR can also be referred to task relocation. However, since task relocation can also refer to the migration of a software task to hardware and vice-versa, a concept termed heterogeneous task relocation [141], we make a distinction here by explicitly referring to the migration of a hardware circuit from one region on the chip to another as PBR or homogeneous hardware task relocation, as different from heterogeneous task relocation.

PBR is a technique of removing a configured circuit from its original location and reconfiguring it in another location on the FPGA and it can be performed offline or in runtime. Runtime PBR is enabled by DPR, allowing runtime access to the configuration memory of the FPGA without power cycling the device, and permitting circuits in damaged chip regions to be configured in another region while the rest of the FPGA remains operational. PBR involves adapting an RM for placement in more than one RP.

Realizing bitstream relocation involves the manipulation of the frame addresses in the task's bitstream before (offline) or during (online) configuration. When done offline, methods used include manipulating the frame address in the bitstream file as seen in PARBIT [142] to generating and storing multiple copies of the partial bitstream for the different target locations. PARBIT is an offline C-based command-line tool for reformatting the original bitstreams generated by Xilinx tools into partial bitstreams targeted for placement in their original locations or different locations. Unlike PARBIT,

in other offline relocation techniques, the same design is added as an RM in multiple RPs so that the design can be configured in more than one RP at runtime.

With respect to reliability, the obvious limitation with the offline relocation methods is that it is impossible to anticipate all the possible locations where permanent damages could occur. In addition, generating and storing bitstreams for multiple target locations increases the external storage size and cost. On the other hand, runtime PBR, which involves the modification of the location address of the task online ensures that only one copy of the bitstream is kept. Furthermore, emergent faults can be easily circumvented given that the requirements for PBR are met. While adapting a tool like PARBIT to run online on a processor in the FPGA is a possibility; however, this would incur a large reconfiguration overhead due to the time taken to run the routine.

For runtime PBR, several methods have been proposed and implemented. In [143], the authors present REPLICA, a hardware *bitstream filter* for runtime relocation. REPLICA implements a subset of the PARBIT functionality but unlike PARBIT, it manipulates the location address online while the bitstream is being downloaded, thus avoiding extra reconfiguration overhead and increased external storage. REPLICA is targeted at Virtex/-E architectures and can only relocate designs that use only CLBs. In the second version called REPLICA2Pro [144], which is similar to REPLICA, the Virtex-II/Pro architecture is supported and the relocation of BRAM and multiplier blocks is possible. However, both REPLICA and REPLICA2Pro are limited to column-wise 1D relocation.

The BiRF [145] is another hardware bitstream filter and it has a general structure which is similar to REPLICA, but unlike REPLICA it is targeted at both Virtex/-E and Virtex-II/Pro architectures, with an improvement in speed of more than 50%. The authors in [145] also introduced BAnMaT Light, which is a completely software-based relocation solution implemented in C.

Several other runtime relocation techniques have appeared in the literature. However, the common theme is the provision of a FAR manipulating functionality, a configuration engine, and a means of recalculating CRC. The recalculation of the CRC checksum is important because the deliberate modification of the FAR value will void the checksum pre-computed by the design tools. In order to maintain bitstream

integrity, a CRC recalculation functionality is needed. It is however, possible to relocate a partial bitstream without regard to CRC recalculation. One can simply issue a CRC reset command (0x30008001 followed by 0x00000007) at the point in the bitstream when a CRC checksum would normally be specified to the configuration logic. As a result, it is possible to have circuit relocation implementations that do not include a CRC recalculation feature although CRC is recommended to be used if the FPGA is in an environment where the bitstream transmission is not prone to error [146].

None of the existing relocation techniques seems suitable for encrypted PBs. As a matter of fact, as far as the authors know, there is not a single work (not even a tool flow) that has considered the relocation of encrypted PBs, especially from the perspective of runtime PBR. With an encrypted PB, existing relocation implementations like those just reviewed would require a runtime decryption of the bitstream before feeding it to the ICAP, incurring a resource overhead for implementing the decryption circuit. A technique where an already configured task is read back and relocated on-chip [147][148] looks attractive for relocating encrypted PBs but there is an initial configuration time which increases the system's overhead. In Chapter 5, this thesis will propose a mechanism that offers a solution with considerably less resource and time overheads.

It is worth noting that apart from WL strategies, which require PBR for achieving uniform ageing; and permanent damage circumvention, which requires PBR for moving a task to a damage-free region, there are other justifications for PBR. One is the reduction of external bitstream storage arising from the fact that multiple RMs can share the same RP and hence, only one bitstream has to be kept.

3.2.5 Requirements for Partial Bitstream Relocation

For a circuit to be relocatable from one location to another on the FPGA, a set of stringent requirements must be met, namely: the matching of the source and destination regions in terms of resource type and relative positions, including the routing structure; the provision of dynamic communication between the relocated circuit and the other circuits on the FPGA; the preservation (or rather, avoidance of) of any static routes crisscrossing the target region; and the provision of clocking.

These requirements apply to both offline and runtime PBR, though mostly to runtime PBR. However, it is relatively easy to meet them offline since the designer is in control of the synthesis. Meeting the requirement of a matching location is eased since the synthesis tool can use the place and route to address a situation in which the target location does not have the resources in the required order. Moreover, all the other requirements are related to general interconnect and clock routings and these are well suited for offline handling and align well with the established PR flow constraints of establishing routes offline. Hence, the following considerations are mostly relevant to runtime PBR.

A. Finding a Matching Target Location

The first requirement to be met for PBR is to find a matching target location. The location to be relocated to must have similar resources (by type, number, and relative positions) as the circuit's original location (see Figure 3.2). The PBs must respect the existing irregularities in the FPGA fabric [149]. These can be inherent to the chip structure, such as clock region boundaries or heterogeneous BRAM and DSP columns, but can also be created by the designer when using PartPins.

This requirement is the easiest to meet as FPGA resources are *generally* tiled in regular repeating patterns. In Figure 3.2, RM2 can be relocated to the indicated matched location because the resources in the target location are of the same type and arrangement as those in RP1. It is salient to state that with newer FPGAs like the 7 series and the UltraScale, the architecture of the chip area is more heterogeneous and irregular. This means it is more difficult to find matching locations on the horizontal plane. Vertical matching locations are easier to come by because there is homogeneity at the column level of all known Xilinx FPGAs.

In general there is almost always a matching location for any circuit on the FPGA, not considering other circuits that may be on the FPGA. However, in reality, other circuits could have taken up potential matching locations, in which case other techniques of relocation like the memoization-based method in [150] can be used, though with a limitation in the data width support, but nonetheless, a promising solution to an impossible situation.

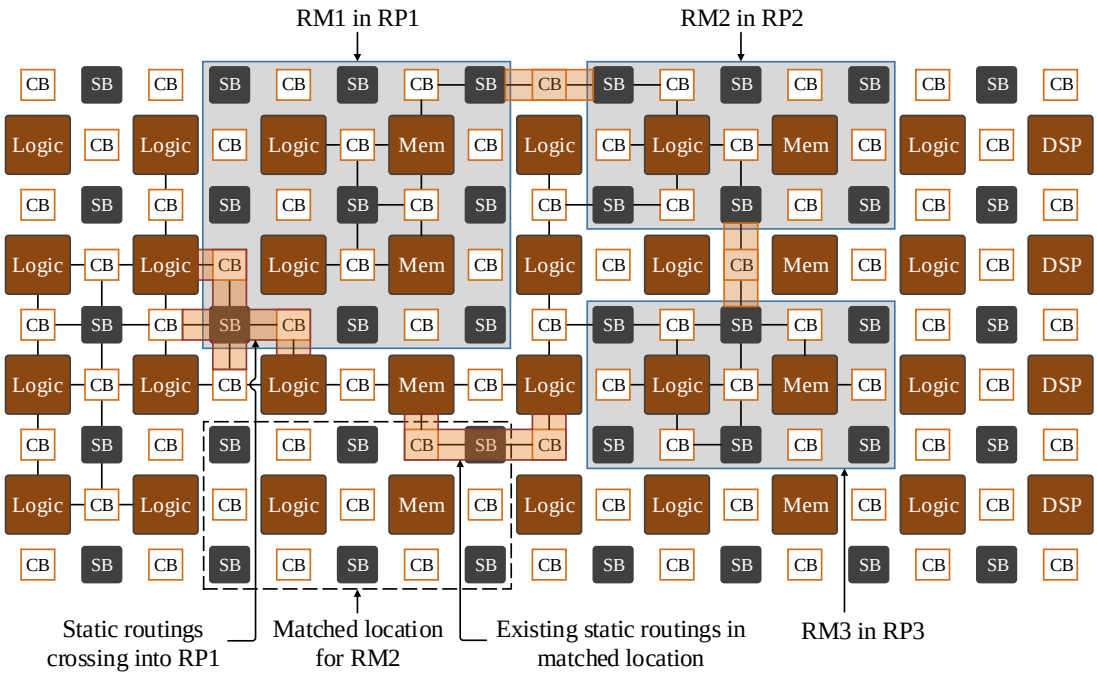


Figure 3.2: Representation of the considerations for circuit relocation

B. Provision of Dynamic Communication

The second requirement that must be met for PBR is the provision of communication at the desired location both for preserving existing interconnections and making new ones. This requirement is generally more difficult to fulfil for runtime relocation because of the need to establish routes in runtime, and is often the question raised against PBR. PBR involves moving a circuit from its original location where it has established communication routes with other circuits on the FPGA. Re-establishing these communication routes after relocation is not natively supported by the existing design tools, as the PR flow demands that all communication routes are statically determined at design time by inserting Partition Pins between the static design and the RMs.

With respect to Figure 3.2, the easiest way to provide dynamic communication is to ensure that routes from RP1 and RP3 to the matched location for RM2 are established at design time. This way, during runtime, RM2 can be relocated while maintaining its communication link with RM1. This can be accomplished by using the PR flow of the vendor which was introduced in Section 2.2.2. The key challenge is that the vendor tool

forces all RMs meant for a particular RP to share the same fixed interface with the static part. The downside of this is that in runtime, an RM can only be placed in or relocated to an RP to which it belongs. Since the decision of which RP an RM belongs is made at design time, this limits the number of locations circuits can be relocated to in case of emergent permanent faults. This deteriorates the reliability figures of the device.

To overcome the limitations imposed by the PR flow, online routing, a technique that involves recomputing the routes through connection and switch boxes by programming the relevant PIPs during runtime can be used, but this is computationally expensive, often requiring several thousands of clock cycles per net [151]. Moreover, determining the location of bits controlling the switch matrices and PIPs in the bitstream is non-trivial and there is no constancy in the bitstream format from one FPGA family to another. With the increasing size of newer FPGA chips, bitstream formats and interconnects have become more complex, and runtime routing inevitably more costly and complicated. Several other approaches have been proposed for dynamic communication to support PBR. These are presented in Section 3.4.4 in the context of on-chip dynamic communication infrastructures in FPGAs.

C. Preservation of Existing Static Routes in Reconfigurable Partitions

One other challenge with relocation is that of existing static routes in the target region which belong to circuits outside that region, and as such must be manually preserved [90]. The target location for a relocated circuit is not guaranteed of being free from interconnect routings that belong to a neighbouring circuit most especially, circuits in the static region. If the relocated circuit is reconfigured in the location without regard to these pre-existing routings, it could override the routings and break the functionalities of the nearby circuits.

For instance, in Figure 3.2, there is an existing static route passing through the matched location for RM2. To avoid conflict, this static route has to be manually preserved when placing RM2. This problem exists because FPGA implementation tools like Vivado allow circuits to use routing resources external to their confined regions even if they have no logic resources there. While the Vivado constraints EXCLUDE_PLACEMENT and CONTAIN_ROUTING can be used to ensure that an

RP does not use logic and routing resources outside its bound, they however, do not prevent routings from the static region from crossing the RPs [37].

A method of preserving pre-existing routings would be to read back the configuration frames corresponding to the routings in the target region and XOR this with those of the relocated circuit before configuration, but care has to be taken to ensure that these pre-existing routings are avoided if their path would conflict with the routing in the relocated circuit. A good allocation algorithmic search should put this into consideration in finding a suitable place for relocation in the first place. One shortcoming of this is the time it takes to perform the readback before configuration and more time could be incurred if this has to be done recursively to find an appropriate location.

FPGA design tools generally allow circuits in the static region to use routing resources in the RPs mostly because of the need to resolve routing congestion. While it is entirely possible to prevent design tools from doing this by using *blockers* that leave the place and route tool with only one option regarding routing a demonstrated in the OpenPR [152] and ReCoBus-Builder (now GOAHEAD) [153] PR tool flows; however, this can exacerbate congestions and increase path latencies [154]. While it is possible to use *blockers*, implementing this is not a trivial exercise and tool support is not yet mature.

D. Timing Constraints Must be Met

One issue that is often overlooked in PBR is timing. When an RM is relocated from its original location, it is possible that it fails to meet timing requirements in the new location. To ease timing closure, it is recommended that inputs in the RM and static logic are registered [37].

Once an RM meets timing within its bounded partition at design time, for the purpose of relocation, the RM can be expected to maintain its timing closure within its own physical bound regardless of wherever it is relocated on-chip in runtime if a strictly matching target location is used. Timing issues are related more to the interface between the static logic and the RM. As the RM circuit is moved about, its physical distance

from the static interface changes and thus its interface timing changes. This timing change must be taken into consideration in a practical PBR implementation.

Timing is addressed in the GOAHEAD PR flow tool by considering the latency on interface signals when generating relocatable bitstreams [154]. Another tool which considers timing issues is AutoReloc [155], which is an automated design flow for generating relocatable bitstreams. To meet timing requirement for PBR, AutoReloc determines the delays between each pin on the RM and the static part for all the RPs to which the RM can be relocated. The maximum of these delays is used to constrain timing for the RM.

E. A Clock Network Must be Provided at the Target Location

The design tools do not route clock signals to regions of the FPGA that are not used by user designs. As such, if a circuit is relocated in runtime, a suitable clock network must be provided at the target location. The clock networks of the FPGA are routed through switch matrices of PIPs. These routing resources can be controlled by activating specific bits in the CMEM. This approach is exploited in [156] to provide online clock routing for relocated circuits. For instance, the authors manipulate regional clock buffers in runtime to provide different clock frequencies to tasks and to also switch away from failed clock networks.

To simplify the process of clock routing, the clock tree routing information that pertains to each RP can be extracted from the static design and added to the RM at design time as done in [152]. Alternatively, dummy primitives like flip-flops or BRAMs can be placed into the floorplan and connected to global clock lines at compile time, forcing the place and route tool to route the required clock network to predetermined regions that will be used for relocation [154].

3.3 Task Configuration in Reconfigurable Computing

The most important service provided by a ROS for RC is hardware task configuration. The capability to dynamically reconfigure an FPGA, brought about by DPR is exploited in RC for swapping tasks in and out of the FPGA in runtime. A review of the existing ROSes in Section 3.1 reveals that DPR is often used. The most popular means of

supporting DPR is through the ICAP. Access to the CMEM can also be achieved through the *Processor Configuration Access Port* (PCAP) [70] in Xilinx Zynq FPGAs as demonstrated in [157]. However, the PCAP can only be controlled from the processor in Zynq FPGAs. As such, though it incurs no hardware resource overhead as per a controller, it is limited to a configuration throughput of 128 MB/s and blocks the processor during reconfiguration, preventing it from running other tasks [158]. Therefore, the better choice is the ICAP when high performance is required.

As shown in Figure 3.3, in a typical ICAP controller, a finite state machine or processor is used to control the ICAP for a variety of read and write operations ranging from task configuration and readback, context switching, PBR, task replication for TMR implementation, and scrubbing for SEM, among others. There is usually a buffer memory composed out of BRAMs and used for buffering or streaming the bitstream. A data mover initiates bitstream transfer from an external storage memory. In order to meet the diverse needs of different RC architectures and ROS implementations, several configuration or ICAP controllers have been developed. These controllers can be broadly classified into two: those targeted at improving configuration throughput and those targeted at reliability.

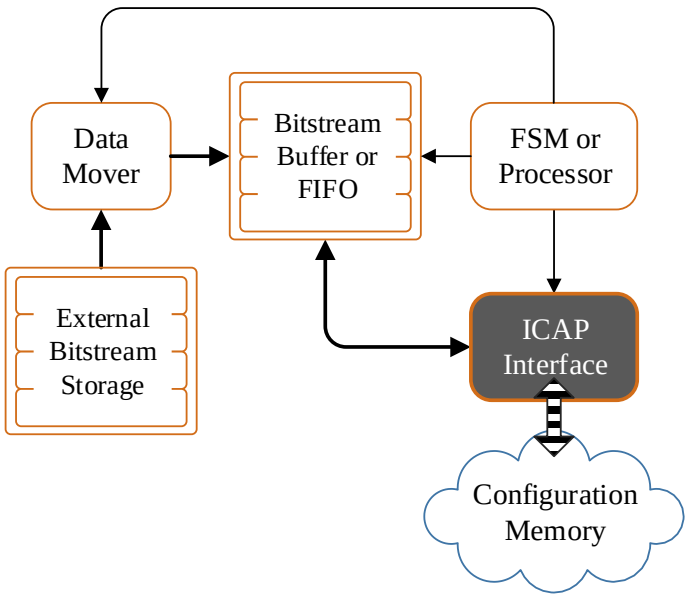


Figure 3.3: Architecture of a typical ICAP controller

One issue with RC in terms of performance is that of reconfiguration throughput especially as multiple system functions compete for access to the same internal configuration interface. A lot of the services provided by a ROS depends on DPR via the ICAP for self-reconfiguration. The maximum bandwidth of the ICAP is 3.2 Gbps at the recommended maximum operating frequency of 100 MHz. There are several attempts at increasing configuration throughput by overclocking the ICAP as seen in [159], [160], [161], and [162], even up to 550 MHz in [163], but this is substantially over the recommended maximum for which the chip has been characterized and guaranteed to perform reliably by Xilinx. Moreover, the maximum overclocked frequency is expected to be device-dependent and even vary among devices with the same speed grade [164] and could lead to erratic behaviours that could be hard to diagnose. As such, overclocking the ICAP would not be a safe option. Even when a feedback approach is used to ensure a continuous correct operation of the ICAP and the device [161][164], it is challenging to guarantee that there is no upset in expected functionality. In fact, the authors of [164] acknowledge the need for a thorough checking to ensure that the functionality of the configured module is correct.

A different way of increasing configuration throughput is the use of bitstream compression enabled by the *Multiple Frame Write* (MFW) feature of the Xilinx FPGA. The MFW command (see Table 2.3 and Table 2.4 in Section 2.1.1) can be used to perform a write of a single frame data to multiple frame addresses. This compression exploits the fact that a bitstream usually contains many configuration frames that are the same. This is especially true for BRAM contents initialized to zeros. This feature is exploited in [165], where the authors notice speedups of up to 4.6x and 2.7x for test relocatable PWM and K-means circuits respectively.

Apart from increasing configuration throughput, ICAP controllers have also been implemented to provide features useful for adaptation. From the viewpoint of a basic task loading requirement in RC, the only feature required in an ICAP controller is task configuration. However, the need for reliability has driven feature expansion in ICAP controllers. Most controllers implement basic features like task configuration and CMEM readback. However, some have added fault-tolerant capabilities like scrubbing and PBR with the purpose of supporting reliability in RC.

For instance, in [164], the ICAP controller features a *Statistics Block* with two hardware counters for system monitoring. The counters keep track of reconfiguration clock cycles and the performance of the controller itself. The values kept by these counters are meant for system-level adaptation management. As a further example, a functionality for modifying LUTs online without the use of precompiled bitstreams is introduced in [166], allowing a time-efficient dynamic modification of logic functions through runtime-generated partial bitstreams for LUTs. This can be a useful feature for error mitigation in LUTs.

The reliability of an ICAP controller itself is important when it is designed for reliable CMEM access. In [167], selective TMR is used, in which a selected component of the controller is hardened by TMR. With this implementation, the controller as a smaller area footprint of 49% of a full TMR implementation. The features of this fault-tolerant controller include hardware task downloading, hardware task blanking, and internal scrubbing in conjunction with the FRAME_ECC primitive. TMR is applied to a special self-recovery module that is aimed at recovering the controller in case of an error by fetching a golden bitstream from external memory. Since TMR is not applied to the controller itself, error detection is done by monitoring the BUSY signal of the ICAP.

A comparison between different fault-tolerant implementations of an ICAP controller is presented in [168]. The authors investigate the reliability and area overheads of using TMR, *Dual-Modular Redundancy* (DMR), and CRC to protect the controller. In the DMR version of the controller, a small TMR-based recovery controller (much like in [167]) is used to recover when the DMR indicates an error. A multiplexer arbitrates access to the ICAP between the main controller and the recovery module.

3.4 Communication in Reconfigurable Computing

On-chip communication architectures can be grouped into three main categories, namely P2P, Bus, and NoC, based on the structure of the physical interconnect, the protocol of data transfer, and the interface design [169]. The main characteristic of P2P

interconnect is the simple and direct interconnection between two communicating circuits, but it is quite inefficient in terms of scalability as the number of cores increases. The shortcomings of P2P architecture scales up in shared buses.

While a shared-bus system allows multiple cores to communicate by granting them access to a central global bus; however, because of the diverse nature and sheer number of these cores, buses become longer, introducing longer communication latencies, and consuming more power [170]. Bus arbitration also becomes more complicated. Buses are not flexible and scalable enough as an addition of a new module requires that the entire system be redesigned. As a result, NoCs have been proposed as the future of on-chip communication.

3.4.1 Network-on-Chip for Communication

The NoC was borne out of the need to improve scalability, modularity, and performance among other factors, in on-chip communication [171]. This need arose because of the increase in the number and type of modules or processing elements running and communicating on a device. CPUs, graphic processors, DSPs, memory elements, and other modules with different functionalities became common-place on a single chip, effectively giving rise to the idea of *System-on-Chip* (SoC).

The deficiencies of dedicated P2P and bus architectures are rooted in the reliance on the routing of wires between communicating circuits. With the increase in networking requirement as more cores are added, wires become long and connections more complicated, leading to increased power consumption. In fact, with the increasing logic resource density of modern FPGAs, on-chip interconnects have become more complex and take a considerable share of a device's power consumption. For instance, in an on-chip interconnect power measurement carried out in [172] on a real hardware, the authors reveal that up to 14% of the total dynamic power can be consumed by the interconnect. It is clear that modern on-chip communication cannot rely on connection-based interconnections.

All these deficiencies have given rise to the notion of routing packets, and not wires [173], which is the main idea behind the NoC. Instead of establishing P2P

connections, whether based on direct dedicated interconnections or shared buses, the NoC abstracts the *Data Link Layer* (data transfer on wired links) from the *Application/presentation Layer* (the on-chip cores). That is, it decouples computation from communication with the potential to bring about unprecedented levels of scalability and performance. The general structure of a NoC is shown in Figure 3.4, with 3-by-3 nodes as an example. NoCs come in various forms targeted at addressing different performance metrics, but in general, they are made up of routers, adapters, and links that connect all the cores (processing elements or hardware tasks) on a chip. Each core is interfaced to the network via a network adapter that implements a network interface on the network side and a core interface at the core side.

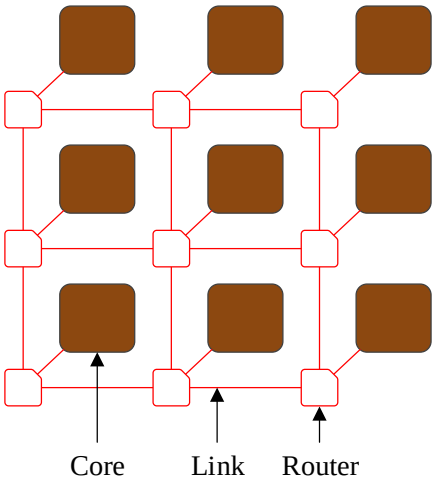


Figure 3.4: Architecture of a generic NoC

In terms of network topology, which is the arrangement and connectivity of the routers, the NoC shown in Figure 3.4 is a typical mesh architecture, which is the most popular due to its support for easy physical design and scalability, but its performance degrades with increasing network size, as it takes more hops for packets to reach their destinations. There are other topologies like *Ring*, *Star*, *Butterfly Fat-Tree*, *Polygon*, and *Torus*. There is no one-size-fits-all topology, and in fact, this is the same for the other NoC design parameters in general, as they are selected based on compromises that deliver the targeted network performance, especially for throughput and latency metrics [174].

One important NoC architectural parameter is routing, which determines how communication packets are routed from source to destination nodes. In a *deterministic* routing, packets are always transferred along the same fixed route unlike in *adaptive* routing where packets can take alternate routes to the destination depending on the network load which is dynamically evaluated [175]. An example of a deterministic algorithm is the XY routing, which moves packets in the horizontal direction first as far and close as possible to the destination node (to a node that has the same X coordinate as the destination node), and then goes in the vertical direction until it arrives at its destination. However, several other routing algorithms have been proposed. A good review of these algorithms has been presented in [176].

There are several other architectural features of NoCs. However, as the concept of NoCs is an already comprehensively covered subject, more extensive details on its basics can be found in [171], [175], and [177]. Nevertheless, it is pertinent to identify the performance parameters of NoCs, which are bandwidth, throughput, and latency [175]. Measured in bits per second (bps), the bandwidth of a NoC is the maximum rate of data transfer and it usually considers the entire packet. Throughput makes allowance for the fact that a packet usually contains non-message-related header and tail information. As such, it measures the rate of transfer of the message payload in messages per second or messages per clock cycle, or normalized to bits per node per clock cycle. Both bandwidth and throughput scale with the number of channels. Latency is the time elapsed from the instance a packet departs a source node to the moment it is completely received at the destination.

3.4.2 Shortcomings of NoCs

As promising as NoCs are, they have their downsides. Though they offer a good communication solution when compared to dedicated P2P and bus communications, there is an attendant resource overhead, which can be significant in smaller devices. That is, NoCs lead to an increase in the footprint of the overall design, and this is due mainly to the additional resources used for the routers to grant network access to the tasks. Depending on the size of the network, an overhead of up to 34.8% (3227 slices) for a 2-by-2 network is not impossible [178]. To reduce an NoC's area utilization, a bit-

serial network access can be used as proven in [179], where in a comparative analysis of serial and parallel interconnects, the authors note significant improvements of up to 5.5× and 17× power consumption and area utilization respectively of serial links over parallel links. Similarly, in [180], the author observe that bit-parallel routers are 8× (for LUTs) and 23× (for FFs) larger than bit-serial routers. In addition, bit-serial designs are noted to have route congestion factors of only 1-2% compared to 10-20% for their bit-parallel counterparts.

Moreover, while compared to shared buses, NoCs lend themselves more readily to runtime circuit placement because of their support for easy modularity and scalability; however, the static routes of the network links still constitute a bottleneck to circuit relocation. In particular, the traditional NoC links pose the challenge of static routes as these links are constructed from the chip's general routing resources and are free to cross RPs in partially reconfigurable system architecture. In a bit-parallel NoC, the network adapter at each node creates static routes that cross into other nodes. A bit-serial NoC that uses general interconnects as links would have lesser static routes, but it would still require online redetermination of route in order to support dynamic communication.

Meanwhile, it is possible to completely do away with routers and still have comparable or better network performance as demonstrated in [181], where a routerless NoC implementation shows a 7.7× reduction in power, a 3.3× reduction in area, a 1.3× reduction in *zero-load* packet latency, and a 1.6× increase in throughput when compared to a router-based NoC.

3.4.3 Bit-Parallel and Bit-Serial NoCs

The interconnections that carry packets from router to router can be made up of several single wires to form a parallel link that is able to switch a multi-bit data at a time. This is the typical case with NoCs and such NoCs can be referred to as *bit-parallel* NoCs. On the other hand, the link can also be composed of a single wire which is able to transmit a bit of data at a time, and as such, the resulting NoC can be termed *bit-serial*.

Although bit-parallel NoCs generally offer higher throughputs, however, it has been shown that a serial implementation has the potential to reduce the area overhead

and power utilization of NoCs [179] while at the same time improving noise and signal interference, offering simpler network layout, and enhancing timing verification. It turns out that because of the efficiency it brings, high-speed serial communication is the current trend in digital design, e.g., PCI Express. As a result of the serial single-wire implementation, the usual performance-limiting skew on parallel links is localized to a single link and as such a much higher frequency is possible with a serial link.

A bit-parallel link can provide a higher throughput than a bit-serial one when clocked at the same frequency. However, in the long run a bit-serial link can achieve a higher throughput if it can be clocked at a fast enough rate, at which point a bit-parallel link fails because of skew. For instance, in [180], the authors demonstrate bit-serial NoC routers that were 2-3x faster than their equivalent bit-parallel routers even with some level of pipeline optimization in the parallel implementation.

3.4.4 The Need for Dynamic Communication

One of the key requirements for dynamic task loading and PBR is the provision of dynamic communication for relocatable circuits. As such, the need for dynamic communication infrastructures is a salient one. In [148], the approach to dynamic communication involves the use of LUTs. LUT-based communication interfaces are attached to the lower right corner of RMs with vertical reconfigurable slots marked out on the chip. These slots have right-aligned *vertical routing channels* with LUT-based communication primitives. This arrangement allows an RM to be dynamically connected to existing RMs on the slot or other slots via a hard-wired macro that runs through the bottom of all the slots. This macro also makes connection to the static region.

A different approach to dynamic communication is taken in DyNoC, a dynamic network-on-chip architecture [182]. While several research works have been carried out on dynamic or reconfigurable NoCs, most do not actually consider the placement of a new task. Rather, they are mostly concerned with the runtime restructuring of the network topology or packet routing to meet changing communication needs as seen in ReNoC [183] and Hoplite [184] respectively. On the other hand, DyNoC's approach to dynamic communication involves placing a new circuit over existing deactivated

network routers while leaving surrounding routers free for communication. With this arrangement, a new circuit can be placed anywhere on the mesh network with continued access to the network. However, we deem this approach to still have the challenges of static routes as the authors do not seem to have provided details on how these are managed and their implementation diagram [182] shows routings crisscrossing the entire floorplan. Indeed, it is unlikely that the authors intend DyNoC to be a communication network for relocatable circuits, as this is not a claim in the work.

An ideal situation for dynamic communication is to have no static interconnects to deal with nor have the need to create routes on the fly. A step in that direction is taken in [90], where the authors present a communication mechanism that involves using the ICAP to transfer data between arbitrarily-placed hardware tasks, in the context of achieving the relocatability of tasks. This is done by connecting memory elements (distributed RAMs or BRAMs) to the inputs and outputs of circuits to serve as data memories and using the ICAP as a side channel to copy data from output memories to input memories thereby avoiding static interconnects. The data contents of a memory element can be accessed online from the CMEM (configuration layer). As such, reading back the correct portion of the CMEM in runtime would give access to the data written by a task into an output memory in the functional layer. The readback data can then be written to the input memory of another task via the configuration layer. Figure 3.5 shows how a task would be interfaced with input and output data memories.

Indeed, the idea of transporting data using the configuration layer of an FPGA dates back to 1998 when Brebner et al. in [185] proposed a virtual communication channel that involves reading from and writing to registers within source and destination virtual tasks defined as swappable logic units. In addition, the work in [160] demonstrates the use of the ICAP for moving data between circuits, but this is not used as a means of providing dynamic communication support with respect to relocation.

This idea of moving data from one task to another without using physical wires can be seen as a form of relocation and it has limitations and consequences as highlighted in [186]. There is no way to know when a task has finished computation apart from polling the task. With multiple tasks possibly simultaneously active, this is even more demanding. There are three operations needed to be performed for each data relocation

– polling, readback, and writing. All these operations have to be serialized since the ICAP is a single resource. That is, ICAP-based data relocation does not support concurrent communication and this is the main bottleneck of transporting data in the configuration layer.

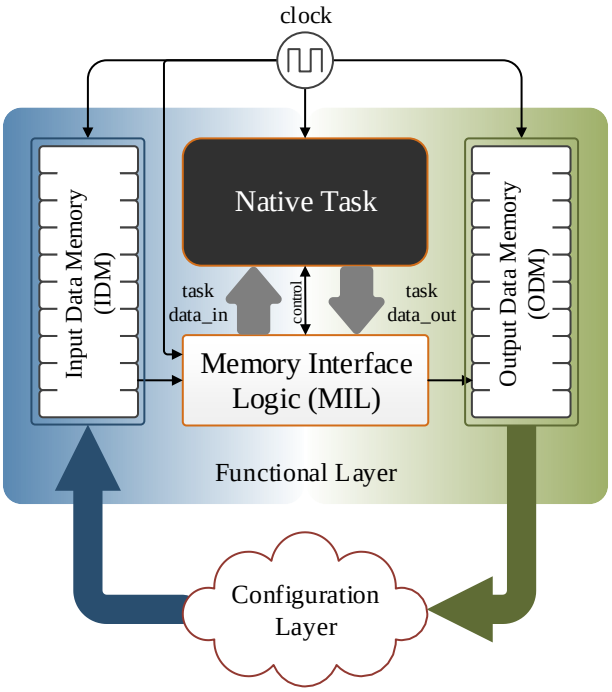


Figure 3.5: Task interfacing for transferring data using the configuration layer

Furthermore, the single nature of the ICAP has an implication on reliability. The ICAP has a maximum theoretical bandwidth of 400 MB/s [37] and Xilinx recommends that more than 99% of this bandwidth should be dedicated to SEM [187] for the entire device. SEM is indispensable for reliable RC. Using at least 99% for SEM means that only a meagre 4 MB/s of the ICAP's bandwidth is available for other functions. With communication drawn in, there are two system functions competing for the remaining 4 MB/s. In other words, time spent on communication is time not available for SEM and configuration.

3.5 Real-Time Systems and Requirements

The concept of real-time computing is often confused as to mean executing tasks instantly. The main objective, on the hand, is to execute tasks before specified deadlines. It is not a matter of how soon or fast, but of when. Even if it takes a hundred years, so far it is completed before its deadline, an execution can be considered real-time. Therefore, in general, "real-time systems are those in which the correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced" [38].

Despite the fact that the emphasis is on meeting deadlines, there are real-time systems in which missed deadlines do not lead to catastrophic system failures. Such systems are referred to either as *firm real-time* or *soft real-time* systems; firm when the delayed results are useless, and soft when the results can still be used. For example, dropping one or two frames in a video processing system due to missed timing would only affect the quality of the video stream, causing some performance degradation and placing this in the firm category. The same cannot be said of the flight control system of an aircraft or an anti-missile system where a missed deadline can have a fatal implication. Systems with such a stringent timing requirement are categorized as *hard real-time*. The categorization of real-time systems into hard, firm, and soft also translate directly into deadline ranges. While hard real-time systems can have time bounds in the order of several microseconds to a few milliseconds, firm real-time systems typically have time bounds in the range of a few milliseconds to several hundreds of milliseconds, and soft real-time systems can cope with time bounds in the range of a fraction of a second to a few seconds [127].

In terms of expected basic requirements, real-time systems are expected to be *timely* – producing correct results both in terms of value and timing; *predictable*, guaranteeing all timing requirements; *efficient*; *robust* – able to retain anticipated functionality even in overload scenarios; *fault-tolerant*; and *modular* – allowing easy modification of system features and functionalities [188].

To account for the mixed-criticality of tasks in a real-time system, where tasks have varying levels of criticality and as such, different strictness of timing requirements, with

some even completely non-critical, priorities can be assigned to these tasks. While missed deadlines in safety-critical real-time task execution can have catastrophic effects; in a bid to keep to deadlines, it is imperative to ensure that other less critical lower-priority tasks in the system are not starved of resources. At the same time, *priority inversion*, a situation where a low-priority task blocks a high-priority task for an unbounded period of time [189], must be avoided.

It would be beneficial to highlight the importance of the awareness of real-time constraints in the key services of reconfigurable computing. This will highlight the factors that have to be considered in the design and implementation of these services,

3.5.1 Real-Time Concerns in Configuration Memory Access

The main real-time concern with CMEM access is with the ICAP and this stems from the fact that the ICAP is a single resource and multiple system functions in a ROS compete for its use. Two key functions that require the ICAP's bandwidth for CMEM's access are task configuration and error mitigation. There might even be multiple task configuration requests simultaneous or in close time proximity requiring access to the ICAP.

The theoretical maximum bandwidth of the ICAP is 3.2 Gb/s at 100 MHz [37] and out of this, at least 99% is recommended to be put towards SEM [187] for effective device-wide coverage. This will affect the responsiveness of the ICAP to reconfiguration requests as much more time is spent on SEM, though scanning the entire device for errors all the time is questionable in the first place and this is the subject of this thesis in Section 4.3.2. Nevertheless, it is evident that there is a need to properly arbitrate the CMEM access in a reconfigurable computing system to ensure that real-time constraints are respected.

From the point of view of command and frame data loading, a task configuration operation through the ICAP is deterministic, especially since there are no external dependencies that might introduce unbounded timing. However, the possibility of configuration errors should be considered as they could necessitate an increase in the overall configuration time due to error diagnosis and bitstream reloading.

In reality, the management of the ICAP as a single shared resource is a real-time scheduling problem and it has to be treated as such. This has been the subject in several research works like [190], where the problems of priority inversion high-priority reconfiguration requests, and that of starvation of low-priority requests are addressed by introducing reconfiguration pre-emption, in which a reconfiguration process can be interrupted and completed at a later time without restarting the data loading from scratch. In another research [191], a reconfiguration controller with a *command-based reconfiguration queue* (CoRQ) was implemented to provide guaranteed reconfiguration latencies and support for the timing analysis of *Worst-Case Execution Time* (WCET) guarantees in the system. WCET is the upper bound execution time of a task and is used for validating real-time systems [192]..

Another factor to consider, which relates to CMEM access is that partial reconfiguration can have an unpredicted impact on resource sharing. For instance, if a block of memory included inside an RM is shared by multiple tasks, the memory will not be available during the reconfiguration operation. It is also possible for the memory to be completely unavailable after reconfiguration if the RM has been inadvertently replaced with one that does not include the shared memory. These are possibilities that are worth considering for real-time PR.

From the viewpoint of secure task (re)configuration using encrypted bitstreams, the restrictions in the port width (only the x8 port is allowed) of the SelectMAP and by extension, the ICAP interface (see Section 2.3.1), means that the configuration time is at least be quadrupled. This increase in configuration latency has to be considered when determining the WCET for a reconfigurable computing system that uses encryption.

3.5.2 Real-Time Concerns in On-Chip Communication

From real-time standpoint, the most important communication metric is predictable and guaranteed bandwidth and data transfer latency [193]. As communication traffic flows in the network from one node to another, there must be a guarantee on the maximum time it would take a packet to reach its destination.

The risk of unbounded latencies is particularly exacerbated in a NoC, where routing algorithms can allow a packet to circulate in the network indefinitely in a bit to

find a non-congested path to the destination, a condition known as *Livelock*. In terms of a packet never reaching its destination, this is similar to *Deadlock*, in which two or more packets block one another indefinitely because needed network resources (e.g., buffers and channels) are fully occupied by other packets. Deadlock is as well detrimental to real-time processing since it is unbounded. Deadlock can be prevented by a careful routing algorithm implementation [170]. The XY routing never experiences either deadlock or livelock because it deterministically routes packets [176]. As such, it ensures predictable latency and is suited to real-time networking. An extensive survey of real-time NoCs can be found in [194].

A challenge with real-time on-chip communication, which arises from DPR, is that there is no way to predict the behaviour of an RM that is undergoing reconfiguration [37]. As a result, if the interfaces between RMs and the static region are not properly *decoupled*, there could be inter-task communication deadlocks where a task waits for a long time expecting data packets to arrive from another task which has been removed from the hardware fabric. This could lead to unpredictable and unbounded communication latencies. Proper handshaking techniques to release RMs safely for reconfiguration would help resolve this potential bottleneck.

3.6 Towards Secure and Dynamic Reconfiguration in RC

Indeed, a lot of research effort has been expended on the development of ICAP controllers as the survey in [195] and [196] show. However, there are important features that have not been given adequate attention in the existing configuration controllers. Since one of the aims of this thesis is to provide configuration infrastructure for reliable RC, ICAP controller features that will support key reliability-enabling functionalities must be implemented. These features include internal register read and write for device diagnosis; CMEM access abort mechanism, and SEM functionalities. These features and more are the subject of the methodologies in Chapter 4.

Moreover, there is a key limitation with PBR with respect to a secure bitstream format. PBR requires the runtime manipulation of the bitstream's frame addresses before it is delivered to the configuration interface. As such, access to the readable

plaintext decrypted bitstream is needed. However, the frame addresses are specified in the encrypted portion of the secure bitstream and are as such, in cyphertext format. Meanwhile, to prevent a breach of security during reconfiguration, the on-chip decryptor in the FPGA feeds the plain decrypted data directly to the configuration interface; and as such, access to the decrypted data is denied. Therefore, there is a motivation to develop a secure bitstream format that is amenable to relocation. This, as well as supporting configuration controller and software for parsing vendor bitstreams are proposed in Chapter 5.

In addition, it is evident that dynamic communication is important for reliability in RC systems and that the static nature of the interconnect wires employed in state-of-the-art inter-task solutions is a big bottleneck, creating relocation-hampering rigidity in the hardware fabric. A motivation, therefore, exists to search for alternate communication solutions that would be amenable to circuit relocation. Chapter 6 presents our approach to dynamic communication, which involves adapting the on-chip clocking resources for inter-communication among relocatable circuits. Since the clock resources use interconnect wires that are independent of the general logic interconnect wires, they do not constitute static routes. In other words, since the clocking resources are in a separate layer (refer to clock layer in Section 2.1.1), communication signals can be safely routed through them, freeing up the circuits to move freely in the functional layer.

3.7 Chapter Summary

This chapter has presented an overview of reconfigurable operating system with particular attention drawn to the various approaches to chip partitioning, reconfiguration style, reconfigurable area style, and inter-task communication. A conclusion that can be drawn is that there is a good body of existing work. Yet, there is a limited reliability or a complete lack of it and this can be attributed to the nature of the inter-circuit communication frameworks used. Even when a whole new approach is taken (e.g., configuration-layer-based data transfer), this falls short in other reliability scenarios. The general reliability concerns in RC were then discussed and it stood out

that partial bitstream relocation is a key technique for enabling reliability, most especially for permanent chip damage mitigation and circumvention.

Furthermore, task configuration and inter-communication methods have been reviewed. Existing configuration controllers were deemed to already be concerned with high configuration throughput and reliability. However, there is still room for improvement, especially regarding reliability-enabling configuration functionalities for COTS reconfigurable devices. Moreover, some implementations went as far as overclocking the configuration interface in a bid to achieve higher performance. However, this approach is counter-reliable as the devices have not been characterized for speeds higher than those reported in the documentation and as such, such solutions cannot be relied upon.

In addition, with real-time processing a consideration in this work, the real-time concerns in configuration memory access and on-chip communication were x-rayed. Predictability, timeliness, efficiency, robustness, and fault tolerance are some of the requirements that have to be met in real-time systems and attention is paid to these in the next chapters where the main contributions of this are presented, especially with the bid to enable high performance, efficiency, and reliability in COTS reconfigurable devices.

Chapter 4

Configuration Memory Access Framework for Reliable Reconfigurable Computing

As noted in Section 3.3, a lot of research effort has been expended on the development of ICAP controllers. However, there are important features that have not been given adequate attention in the existing controllers. For instance, if the system controller in a ROS issues a task configuration command and before the bitstream has been fully loaded, the controller deems it fit that the configuration is no longer needed, it should be possible to stop the operation and reset the ICAP without causing any lockups. Indeed, the ICAP has an *abort* feature that can be used to implement such a functionality. Moreover, as task configuration and SEM both require the ICAP's limited bandwidth for their operations, it is important to ensure that an optimal operation strategy is used to ensure a fair sharing of the ICAP between these two critical system functions without starving any of them. Furthermore, access to device registers for device-level information and control would be handy, especially for system diagnosis and SEFI mitigation. As such, this thesis proposes an ICAP controller with these functionalities. To manage the various low-level operations of this controller is abstracted by a *CMEM Access Manager* (CAM) that incorporates *Direct Memory Access* (DMA) access, operation parameter setting, and operation control.

The techniques and implementations reported in this chapter are covered in the author's publication in [197]:

- **A. Adetomi**, G. Enemali, and T. Arslan, 'A Fault-Tolerant ICAP Controller with a Selective-Area Soft Error Mitigation Engine', in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 192–199.

4.1 Configuration Memory Interfacing

Figure 4.1 is a diagrammatic representation of the proposed CAM. The core of the manager is a *Finite State Machine* (FSM) that interfaces to the ICAP. The ICAP FSM (IFSM) controls the loading of bitstream command packets to the ICAP. A BRAM-based buffer is used to temporarily store the bitstream inside the chip before it is sent to the ICAP. There are also functional blocks for configuration monitoring, FAR command packet detection and frame address modification for PBR.

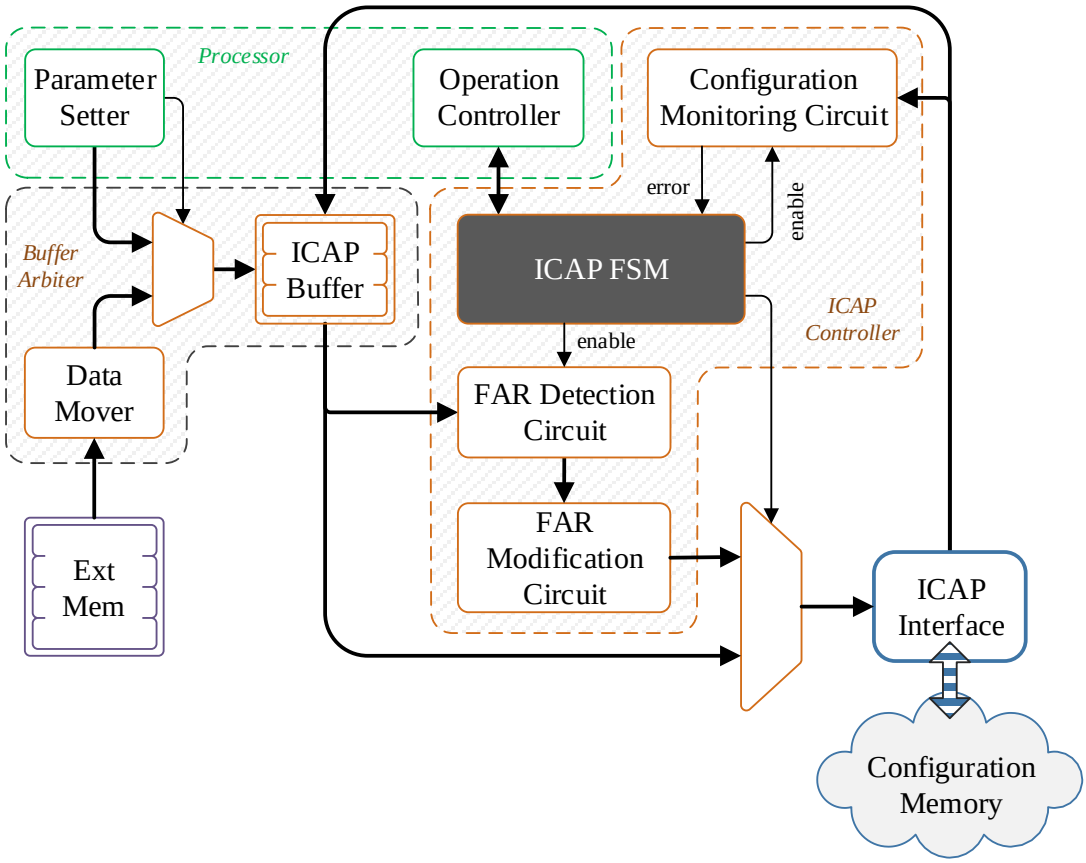


Figure 4.1: Top-level view of the configuration memory access controller

To control all these functions, a controller or processor external to the IFSM is required. In this prototype, the processor used is the ARM Cortex-A9 for the Zynq FPGA and MicroBlaze processor for the 7 series FPGA. The processor is not part of the configuration controller; it is a user-level component. On receiving a DMA transfer request (through an interrupt pulse generated by the IFSM) the processor initiates the

transfer by writing to the control registers of the Data Mover. The following subsections throw more light on the implementation details of the controller's components.

4.1.1 ICAP Controller

The ICAP controller encompasses an FSM which directly interfaces to the ICAP and performs low-level read and access operations. Configuration error monitoring is provided by a dedicated *Configuration Monitoring Circuit* (CMC) while the incoming bitstream is monitored to detect and modify FAR values for PBR operations.

The *ICAP Finite State Machine* (IFSM) is the core of the ICAP controller and it exposes a number of interface ports to the user or a host processor for controlling the FSM. There are other ports that connect to the ICAP primitive and the ICAP Buffer. The IFSM is based on a one-hot FSM encoding, which is better at error detection than binary encoding [198]. Standard FSM coding techniques are followed in the implementation of the IFSM. For instance, an enumerated type containing all possible state values is used to declare the state register. Since an SEU in a flip-flop can throw an FSM into a non-existent deadlock state [102], the IFSM includes a default state that catches all invalid state transitions and passes control to an ERROR_PERSIST state, which asserts the *icntrlr_err* signal. In addition, the Vivado *safe_implementation* and *safe_recovery_state* attributes are used to implement an FSM that is as reliable as possible.

For state control, *state* and *state_after* internal signals are used to simplify the control of an otherwise complex but efficient state machine. To prevent code bloat and unnecessary hardware resource usage as multiple FSM states send commands and data to the ICAP at different points, every write to the ICAP is handled in a WRITE_TO_ICAP state. A correct return of control to the proper state is then achieved by setting *next_state* to the content of the *state_after* register, which is always set before control is passed to the WRITE_TO_ICAP state. This coding style is justified by the fact that the eventual resource utilization is much lower than state-of-the-art considering a similar feature set or even more (see Section 4.5.1).

Figure 4.2 shows the key interfaces and ports of the IFSM. Note that internal interconnection ports like the ones controlling the CMC are not shown. Brief descriptions of the control interface ports are given in Table 4.1.

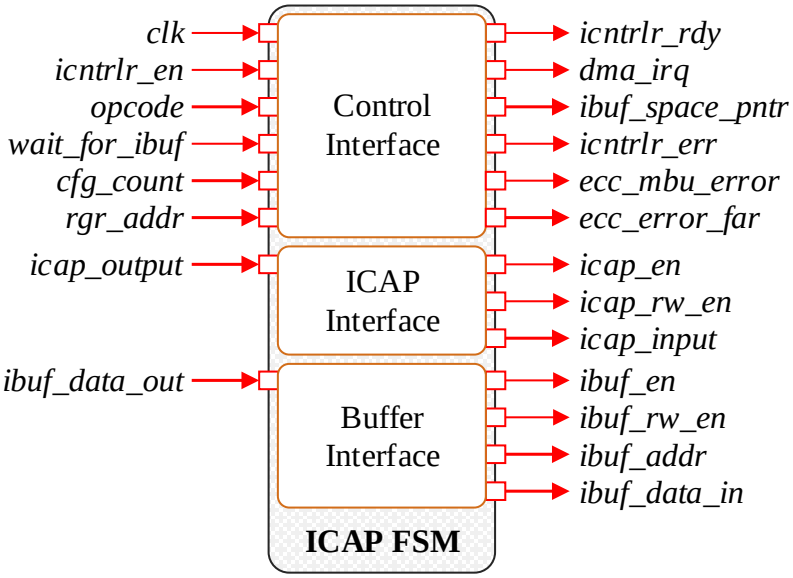


Figure 4.2: Key interfaces and ports of the ICAP finite state machine

In terms of control flow (see Figure 4.3), the IFSM remains in the RESET state waiting for the opcode to change, at which point it deasserts the *icntrlr_rdy* signal and move on to the CHOOSE_OP state. For most of the operations, command sequences and templates are kept in the IBUF and used as the basis for the operations (see Section 4.1.3). Thus, the first step in performing an operation is to point the IBUF address pointer to the appropriate address in the memory. In the CHOOSE_OP state, the FSM first synchronizes the ICAP by writing the operation starting command sequence (see Appendix A.1) and then proceeds to perform the operation chosen by the user. It should be noted that in the implementation, there are multiple distinct states used to perform each operation. For instance, to perform a relocation, the target location address has to be first retrieved from the ICAP Buffer in a different state prior to writing the frame data.

After the operation has been performed, the END_OP state is used to send desynchronization commands to the ICAP while a subsequent COMPLETE_OP state asserts the *icntrlr_rdy* signal and waits for the user's acknowledgement (setting opcode

to NOP) before returning control to the CHOOSE_OP state. Every operation is ended with a DESYNC command to ensure that each operation starts from a known state.

The IFSM's interface to the ICAP is presented in Figure 4.4. Every data word presented to the ICAP's input and read from its output is bitswapped at the byte level. That is, every byte is swapped such that the MSB becomes the LSB. Reading the ICAP requires deasserting CSI_B (= '1') first, and then setting RDWR_B to '1'. Asserting CSI_B (= '0'') after this makes readback data available three clock cycles later. Writing the ICAP is similar except that RDWR_B is set to '0' and the data written one word per clock cycle after asserting CSI_B.

Table 4.1: Description of the IFSM's control interface ports

Port	Description
clk	Clock input to the IFSM. The maximum allowed frequency is 100 MHz
icntrlr_en	1-bit active LOW enable for the ICAP controller. The *opcode* port should be set prior to asserting this.
opcode	4-bit operation selection port. This should be set before asserting *icntrlr_en*
wait_for_ibuf	1-bit signal instructing the IFSM to stall because there is no new data to load from the IBUF
cfg_count	2-bit configuration count port, used to specify how many frame locations should be written for MFW
rgr_addr	5-bit register address port. This address can be for any of the device configuration registers in Table 2.3
icntrlr_rdy	1-bit output port that indicates the completion of an operation and the readiness to start a new one
dma_irq	1-bit DMA interrupt signal to the user for requesting frame data transfer from the external memory into the IBUF
icntrlr_err	1-bit output port to indicate internal FSM error. This is asserted in the ERROR_PERSIST state
ecc_mbu_error	1-bit output indicating that a multi-bit ECC error has been detected in the frame pointed to by the *ecc_error_far* port
ecc_erorr_far	26-bit output indicating the frame address where an ECC error has been detected

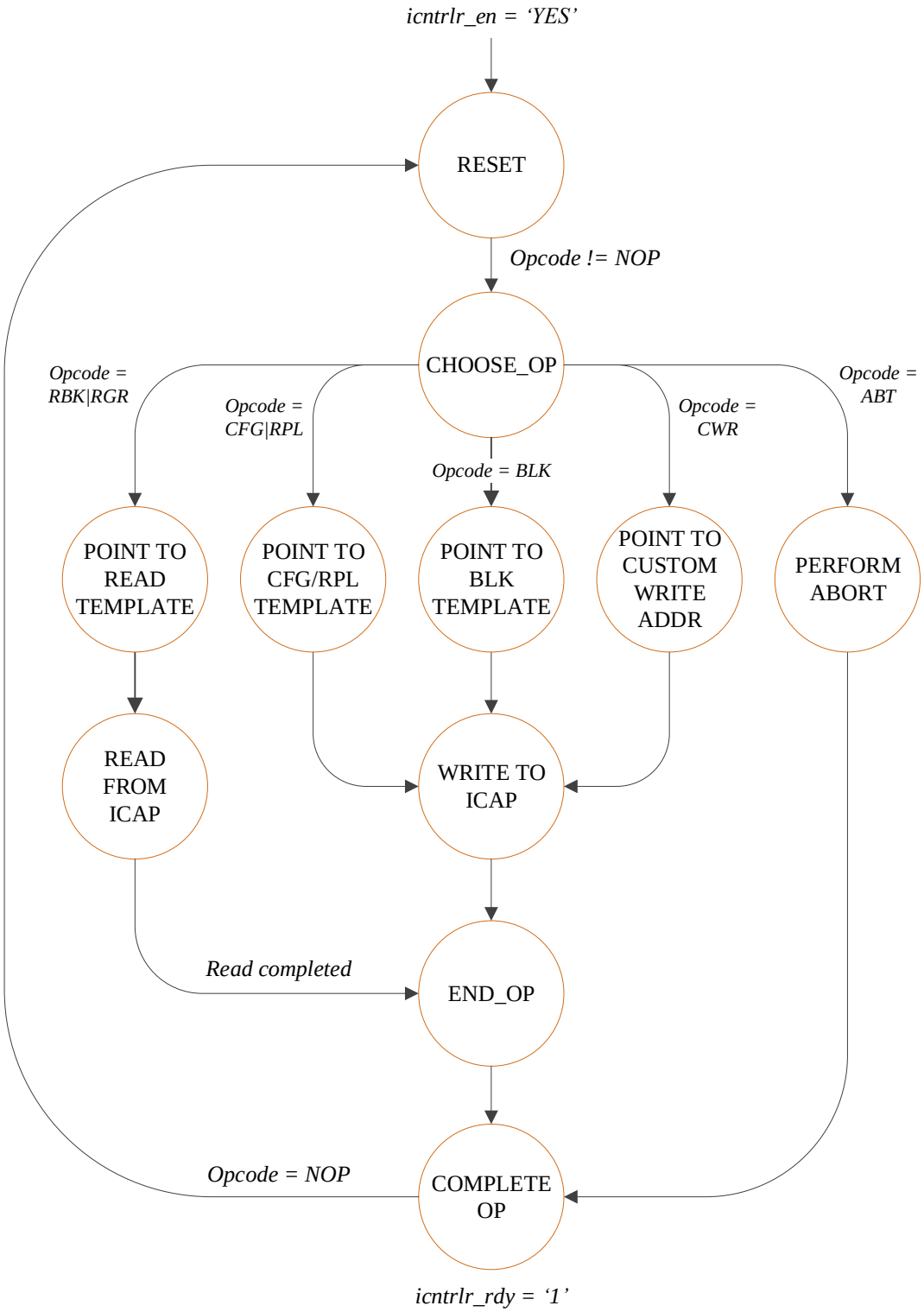


Figure 4.3: State diagram of the IFSM

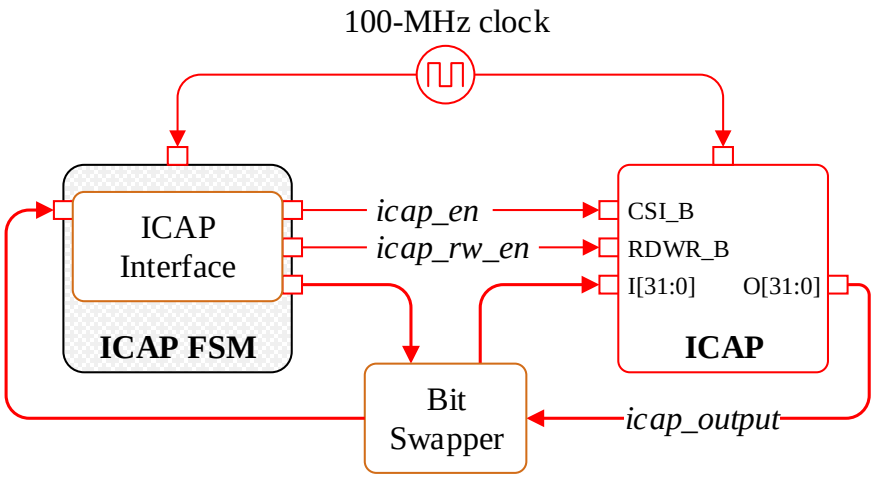


Figure 4.4: IFSM's interface to the ICAP

For PBR operations, the *FAR Detection Circuit* (FDC) and *FAR Modification Circuit* (FMC) work together to detect and modify frame FAR addresses on their way to the FPGA's CMEM. Once a FAR command packet header (0x30002001) is detected by the FDC, the FMC is activated to compute a new FAR value based on user specification. The user is able to control the generation of new FAR values by writing parameters into appropriate IBUF locations (see Section 4.2). Up to three new frame addresses can be modified, and this for TMR implementation purpose. The frame address modification involves changing column, row, and top_bottom values when necessary.

4.1.2 Bitstream Buffering

The partial bitstreams of hardware tasks are stored in an external DDR memory and transferred to an on-chip BRAM-based *ICAP Buffer* (IBUF) by a Data Mover. All these components in addition to the multiplexer at the input of the IBUF are grouped as the *Buffer Arbiter* (see Figure 4.1). In addition to the task bitstreams coming from the external DDR into it, the IBUF is also used to store configuration command templates and sequences, which are merged with configuration data prior to being written to the FPGA's CMEM. The templates require a user to specify certain parameters needed for some operations and sequence are often used commands kept inside the IBUF (e.g., the commands for synchronizing and desynchronizing the ICAP).

The 7 series FPGA's BRAM is dual-ported, with two possible RAM modes [199]. When configured in the true dual-port mode, it provides two independent read/write ports to the user. That is, both ports can access any memory location at any time. On the contrary, in the simple dual-port RAM mode, one port is dedicated to writing and the other to reading. For access conflict avoidance, certain restrictions are recommended, depending on whether the dual-port access is synchronous or asynchronous. This mostly involves using the READ_FIRST, WRITE_FIRST, and NO_CHANGE modes to control which operation (read or write) has priority for each of the ports. This work uses the WRITE_FIRST mode with a true dual-port RAM selection.

As conflicting simultaneous access to the BRAM can cause data uncertainty, the IFSM never accesses the IBUF unless an operation is going on and the user processor should also not set parameters while the IFSM is active. The expected order of controlling of operation is to set parameters in the IBUF first, then start an operation. It is however, necessary to access the IBUF during an active frame data loading, for transferring bitstream data from the external memory but care is taken to avoid access collision by monitoring the *ibuf_addr_pntr*. A *dma_irq* interrupt signal is routed out to the configuration controller's interface for this. The IFSM tracks the location of the *ibuf_addr_pntr* at a frame (404 bytes) granularity level. Once at least a frame space is available, the *dma_irq* is pulsed for one clock cycle. The user can monitor an *ibuf_space_pntr* port for an indication of the position of the *ibuf_addr_pntr*. The frame data buffer space can accommodate 9 frames. Therefore, *ibuf_space_pntr* is a 4-bit signal. The DMA interrupt is fired each time *ibuf_space_pntr*.

The combination of an interrupt request with coarse frame-level information about the space available in the IBUF allows flexibility in data transfer and ensures that the minimum possible delays are incurred, if any. It should be noted that the ICAP clocking is done at the recommended 100-MHz speed. This leaves room for clocking the Buffer Arbiter at a higher rate. If for any reason, frame data is not loaded as at when signaled by the *dma_irq*, the user can assert a *wait_for_ibuf* signal to instruct the IFSM to stall while new frame data is being loaded.

There are three components of the configuration controller that require access to the IBUF, namely, the IFSM, the Processor (to set operation parameters in the IBUF), and the Data Mover. The IFSM is given a dedicated port on the IBUF while the Processor and the Data Mover both have a multiplexed access to the other port.

While the BRAM can practically be of any size depending on how much buffering is desired, a single BRAM36k is used in the demonstration. This provides a total of 1024 32-bit memory locations. An address map of the composition of the IBUF is shown in Figure 4.5. The frame data buffer space is the section used for bitstream buffering and it is initialized to all zeroes. It can buffer up to 9 frames. The following subsections provide details on the contents of the sequences and templates kept in the IBUF.

4.1.3 ICAP Access Command Templates

For the various operations of the ICAP controller, configuration command packets are kept inside the IBUF as templates and are written to the ICAP along with the configuration frame data. The commands are based on the registers in Table 2.3. Details on how to compose read and write commands packets have been presented in Section 2.1.2 and be further explored in [47]. The benefit of using templates is that the commands are generic and thus reusable, minimizing external storage requirement as incoming bitstreams do not have to include the commands already in these templates. The details of the composition of these templates can be found in Appendix A.

4.1.4 Execution and User Interface Flows

The execution and user interface flows of the ICAP controller are shown in Figure 4.6 and Figure 4.7 respectively. At the completion of an operation, which is signalled by the assertion of the *icntrlr_rdy* signal, the Processor or a user-determined driver must change the opcode to NOP in order to revert to the RESET state for a new operation (if any) to start. The RESET state resets the internal registers and it is important for this state to be used intermediate of operations. For instance the *icntrlr_rdy* signal has to be de-asserted in readiness for synchronizing a new operation.

Note that an exception to the user interface flow is the abort operation which can be specified at any time during an active operation. A special circuit in the same process as the IFSM is used to continuously check if the opcode becomes ABT at any point during an active operation.

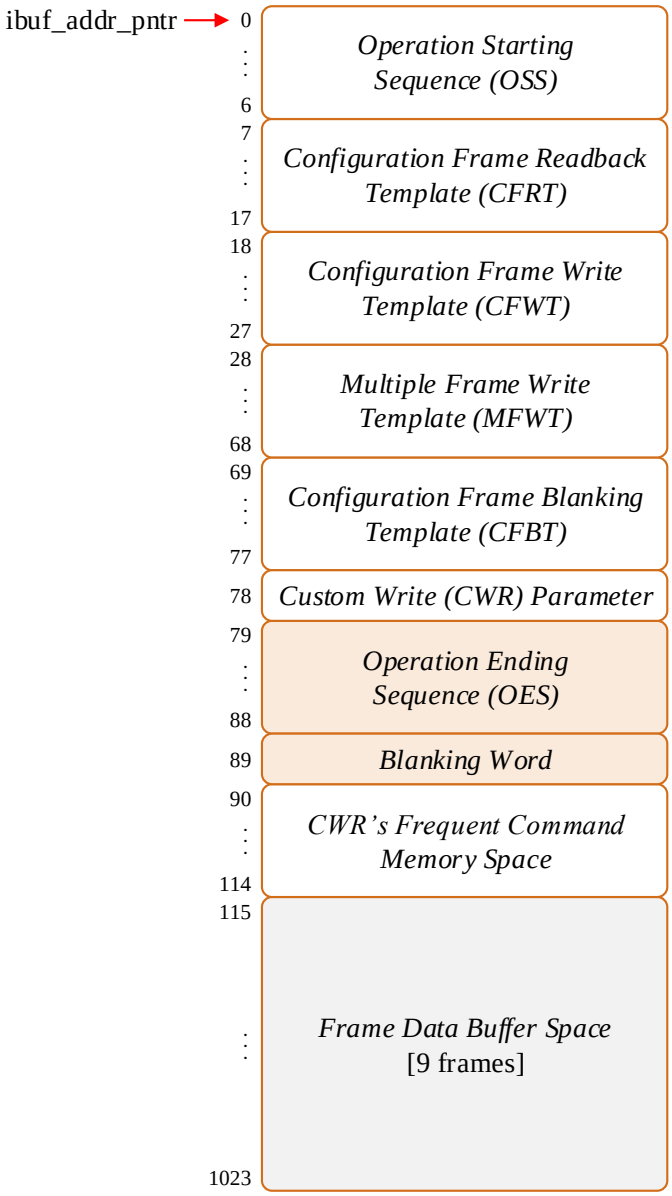


Figure 4.5: ICAP Buffer's memory address space allocation

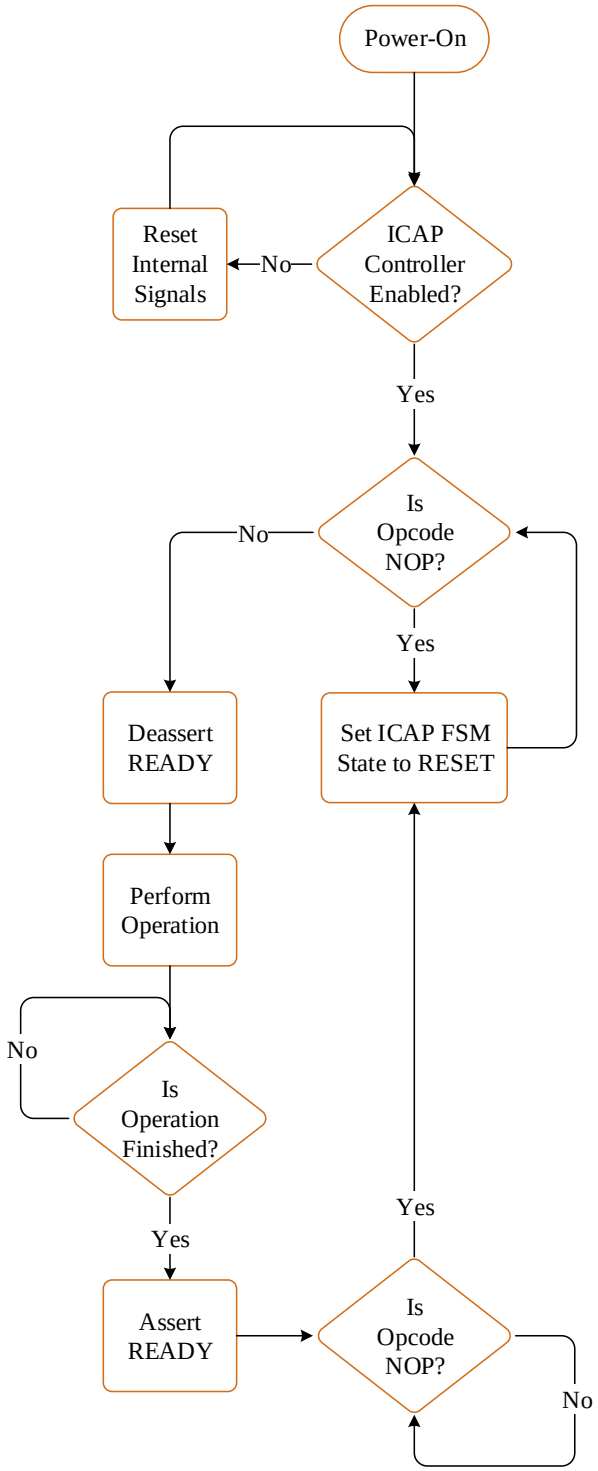


Figure 4.6: ICAP Controller's execution flow

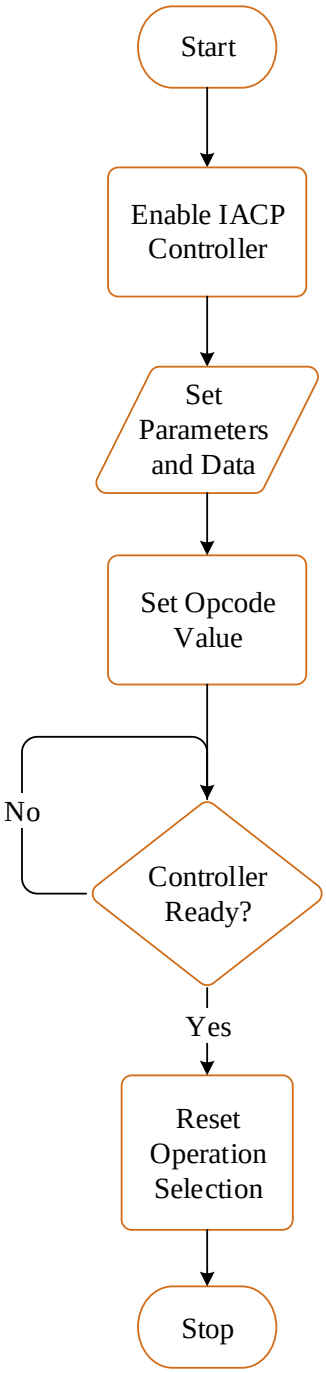


Figure 4.7: ICAP Controller's user interface flow

4.2 Configuration Memory Access Operations

There are eight operations in all. These operations are described in the subsections that follow. A unique operation code (opcode) is assigned to each operation as shown in Table 4.2. This allows a user to specify the operation to the IFSM by writing a 4-bit

code to the opcode port of the ICAP controller. The three most basic templates are the CFRT, CFWT, and MFWT templates and these are used in various forms by the operations. These templates give primitive access to reading and writing the CMEM.

In addition, Appendix B contains relevant waveforms for the different operations. These waveforms were captured during real tests of the operations.

Table 4.2: ICAP Controller's operations and opcodes

Operation	Abbreviation	Opcode	Templates
No Operation	NOP	0x0	None
Readback	RBK	0x1	CFRT
Configuration	CFG	0x2	CFWT, MFWT
Read Modify Write	RMW	0x3	CFRT, CFWT
Blanking	BLK	0x4	CFBT, MFWT
Register Read	RGR	0x5	None
Custom Write	CWR	0x6	None
Abort	ABT	0x7	None
Soft Error Mitigation	SEM	0x8	CFRT, MFWT

4.2.1 No Operation (NOP) – Opcode 0

This is the idle or default state of the ICAP Controller. In this state, no operation is carried out. The implementation of the controller is such that every operation must begin and end with the NOP as shown in Figure 4.6. This ensures that every operation starts from a known state by triggering a reset of the internal signals.

4.2.2 Readback (RBK) Operation – Opcode 1

This operation is used to read back configuration data from the CMEM. CMEM readback is an important functionality used for SEM, especially readback scrubbing (see Section 3.2.2). Each RBK operation retrieves a frame of pad data (zeroes) before the actual configuration frame data, incurring an overhead of 101 clock cycles. At the recommended ICAP clock frequency (f_{ICAP_MAX}) of 100 MHz [49], this is 1.01 µs for

each readback. This overhead is unavoidable but can be amortized over many contiguous frames if many frames are read in a single operation. The most important function of readback is to read the CMEM content for soft error check. The RBK operation requires the user to set the FAR address for readback and the number of words to read, pad frame inclusive (see Table 4.3).

Table 4.3: RBK operation parameters

Operation	Parameters	IBUF Address	Comments
RBK	RBK_FAR	14 (0x00E)	The number of words excludes the pad frame
	NUM_OF_WORDS	16 (0x010)	

The RBK operation uses the OSS sequence, followed by the CFRT template, and concluded with the OES sequence. Before the OES is loaded, the ICAP is switched to the read mode and readback data is available deterministically three clock cycles after asserting *icap_en*. The read-back data is stored in the IBUF from address 0x073. Since the frame data buffer space is only 9 frames, it means only 9 frames can be read in a single operation. The pad frame is discarded and never stored in the IBUF. It is certainly possible to read more than 9 frames in a single operation if the IBUF is implemented to use more BRAMs. The read-back data can, as well, be transferred from the IBUF to an external memory (if necessary) to free up the IBUF.

4.2.3 Configuration (CFG) Operation – Opcode 2

The CFG operation is for loading tasks' bitstreams to the CMEM. This operation intrinsically makes provision for circuit relocation; that is, every configuration request is treated as a relocation request. There are two modes for the CFG operation – *Basic CFG* and *MFW-based CFG*. The Basic CFG uses the sequence and template combination of [OSS + CFWT + Uploaded Data (multiple frames) + OES] while the MFW-based CFG uses the combination of [OSS + CFWT + Uploaded Data (single frame) + MFWT + OES].

Up to three target frame addresses (see Table 4.4) can be supplied when using the MFW-based CFG for configuration, in which case the configuration data has to be

loaded frame-by-frame. Because it uses the CFWT, the Basic CFG incurs a pad frame (see Appendix A.3). This is not the case for MFW-based CFG and as such, it is more suitable for TMR and proves to be more efficient for small-sized tasks. The TMR configuration of a task can be more efficient when the MFW-based CFG is used, in which case the task can be duplicated or triplicated in a single configuration operation. Otherwise, the Basic CFG mode is used to configure the same task in multiple CFG operations.

Table 4.4: CFG operation parameters

Operation	Parameters	IBUF Address	Comments
CFG	DEVICE_IDCODE	25 (0x019)	If MFW-based TMR is not needed, the number of FARs (indicated on the *cfg_count* port) can be ignored
	CONFIG_FAR_1	29 (0x01D)	
	CONFIG_FAR_2	54 (0x036)	
	CONFIG_FAR_3	68 (0x044)	

If there is no intention to use the MFW feature, only the CONFIG_FAR_1 address has to be supplied and *cfg_count* should be disregarded. This is because, the IFSM automatically detects the number of frames to write as the bitstream is being uploaded to the ICAP and automatically determines whether to use the Basic CFG or MFW-based CFG. When the number of words is more than a single frame (101 words), then it knows the MFW cannot be used and it ignores CONFIG_FAR_2, CONFIG_FAR_3, and the value on the *cfg_count* port.

The CONFIG_FAR addresses in Table 4.4 are actually offset addresses. For configuration or relocation to work successfully, the task is expected to be initially floor-planned during the design phase, at the top-leftmost corner of the bottom half of the FPGA (see Figure 4.8). Column and row offsets from this location are then stated in their respective fields in the CONFIG_FAR addresses. In Figure 4.8, N is chosen as -1 so that the first row has Y=0.

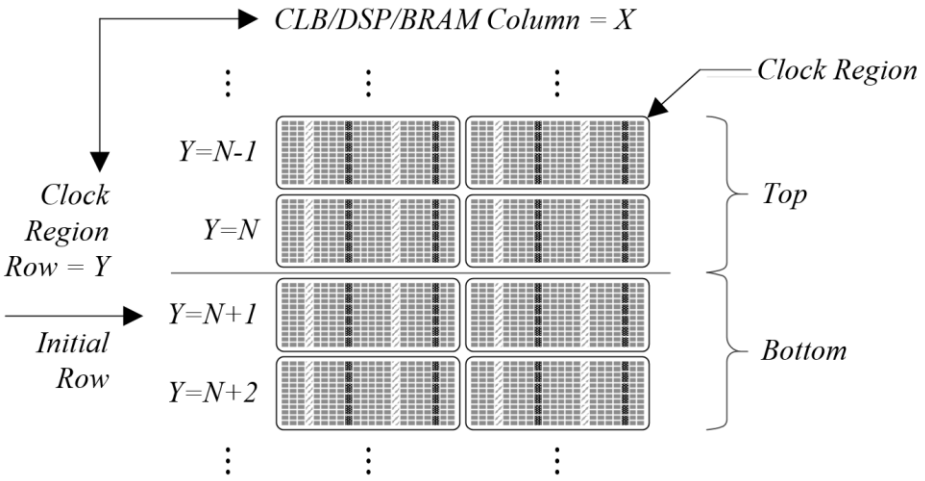


Figure 4.8: FPGA row and column addressing for task locations

4.2.4 Read Modify Write (RMW) Operation – Opcode 3

The RMW operation combines the RBK and CFG operations for a highly-abstracted readback and configuration in a single operation. This is a useful feature for task and data relocation. For instance, a method of PBR is to read back an already configured task and configure it somewhere else on the chip in runtime as demonstrated in [147] and [148]. Once a task is relocated (by any means), it might be necessary to save the context of the task and restore it by copying changeable user memory data in FFs, LUT-RAMs, SRLs, and BRAMs from the original location to the new location. To ensure that relocated task contexts are not lost, the circuit in the original location should not be blanked until the data has been relocated as well. The RMW operation uses the template combination of [OSS + CFRT + Optional Save_Data Bit Toggling + MFWT + OES]. As such, source and destination frame addresses are set in the respective templates.

A special step is taken when a BRAM content frame is being *RMW*ed; there are *save_data* bits in the frame and these have to be toggled to enable the BRAM contents corresponding to that frame to be updated. There is a save_data bit is at bit position 17 of the 5th word in every successive 10 words in every BRAM frame. When the frame is read back, the bits show up as OFF (‘1’). The IFSM turns them ON (‘0’) while writing the frame read-back frame data to the target frame address. This is done without stalling the ICAP.

4.2.5 Blanking (BLK) Operation – Opcode 4

The BLK operation is useful for removing an already configured circuit. A circuit can be removed to reduce power consumption if it is no longer needed. To blank a circuit region, a normal black box bitstream generated by Vivado can be used, but for each circuit configured on the FPGA, a blanking bitstream like this would have to be kept in memory, thereby increasing the size requirement of the external memory. On the other hand, the BLK operation simply requires the starting frame address and the number of frames to blank.

Since a blanking bitstream is composed basically of zeros, an iterative writing procedure is used by the ICAP FSM. Once the user specifies the START_FAR value and the NUM_OF_FRAMES at the addresses indicated in Table 4.5, the IFSM automatically generates the other FAR values. In addition, the BLK circuit utilizes the MFW feature to reduce configuration time. Basically, we write multiple addresses with a single frame of zeros, thereby saving a huge time. In order to reduce logic resources used, the BLK uses the CFG operation's circuit for the MFW part. That is, the BLK operation shares IFSM states with the CFG operation. A BLK operation uses the sequence and template combination of [OSS + CFBT + 101 Blank Words + MFWT + OES].

Table 4.5: BLK operation parameters

Operation	Parameters	IBUF Address	Comments
BLK	DEVICE_IDCODE	70 (0x046)	The number of frames includes the starting frame
	START_FAR	74 (0x04A)	
	NUM_OF_FRAMES	77 (0x04D)	

4.2.6 Register Read (RGR) Operation – Opcode 5

The RGR operation provides an easy access to the FPGA's internal registers. It uses the sequence and command combination of [OSS + RGR command + Readback Event + OES]. A user can set the 5-bit address (see Table 2.3) of the register to read on the *rgr_addr* input port of the CFG Controller and set the Opcode to 4. The ICAP FSM

reads the register and returns the value to the IBUF_DATA_ADDR address. The RGR capability can be utilized to diagnose the FPGA in runtime. For instance, if a configuration error occurs, the STAT register can be queried to know the source of error. For writing a register, the CWR operation can be used.

4.2.7 Custom Write (CWR) Operation – Opcode 6

The CWR operation allows the user to write runtime-generated data into the configuration memory. Since the CFG operation works with the configuration data stored in an external memory and moved into the IBUF, it does not directly support writing data generated at runtime. The CWR is the solution to this. Data buffered on the IBUF from address 0x05A (see Table 4.6) can be written to the configuration memory. The number of words for the CWR is set on IBUF address 0x04E.

Table 4.6: CWR operation parameters

Operation	Parameters	IBUF Address	Comments
CWR	NUM_OF_WORDS	78 (0x04E)	Allows the loading of dynamically-generated data
	CONFIG_DATA	90 (0x05A)	

The CWR operation can be used for device diagnosis by allowing the user to load dynamic data and indeed, this can be more efficient than loading data from an external memory, especially for small chunks of command packets. From address 0x05A, 25 memory locations are reserved for the user to keep frequently-used command words (see Figure 4.5). The memory locations are just before the IBUF_DATA_ADDR (0x073) and are never overwritten by the IFSM. The CWR can also be used in conjunction with the RBK operation for fault injection. A frame can be read back and stored in the IBUF. A bit of interest can then be flipped and the frame written to the CMEM using the CWR operation.

The number of words being written in a CWR operation could also be monitored for confirmation of alignment to a frame of configuration words. However, this is not necessary as any incomplete loading would trigger a configuration error which would be picked up by the CMC. Nevertheless, the user needs to ensure the correct loading of

commands and that frame data words are properly aligned to 101 words. For the CWR operation, the template usage is as follows: [OSS + User-Generated Commands & Data + OES].

4.2.8 Abort (ABT) Operation – Opcode 7

Though the ICAP interface has the capability to abort an ongoing operation, to the best of our knowledge, this has not yet been fully explored and reported in the literature. This capability ensures that if an ongoing operation is deemed to be no longer required due to a change in the dynamics of the system, the system does not waste precious ICAP bandwidth completing the operation. The operation can be immediately stopped and a new one started. The readback abort capability is used in the SEM operation (see Section 4.3.2) to stop readback when a soft error is detected. An abort is triggered by changing the *icap_rw_en* signal while the *icap_en* signal is asserted (= '0'). For readback, abort ends when the *icap_en* is deasserted, while for configuration, it ends after four clock cycles. Note that the ABT operation does not follow the execution and user flows of Figure 4.6 and Figure 4.7. The ABT can be specified on the *opcode* port at any time during an active operation. It can as well be triggered internally by the IFSM.

4.3 Support for Error Mitigation

The diagnosis and correction of errors is an important feature that should be supported by an ICAP controller. Depending on the nature of error mitigation sought, the implemented operations can be used for both SEM and HEM. The CFG operation can be used for blind scrubbing while the combination of the RBK and CFG operations can be used for readback scrubbing of the CMEM. For HEM the PBR functionality offered by the CFG operation can be used to relocate a task to a damage-free region in case of hard errors affecting its original partition. The 7 series FPGA also has an internal Readback CRC circuitry that simplifies readback scrubbing (see Section 3.2.2). The Readback CRC requires access to the ICAP but only when the user design has relinquished control by issuing the DESYNC command. However, the user has no

control on the selection of frames or chip area for scanning. The following subsections present the SEM features implemented by the configuration controller.

4.3.1 Readback Scrubbing Support – SEM Operation (Opcode 8)

The CFG operation can be used by a user-determined scrubber to perform blind scrubbing. However, for readback scrubbing, a distinct operation (SEM operation) is implemented to simplify the process. For the SEM operation, the configuration controller uses the FRAME_ECC primitive in conjunction with the RBK and CFG operations. According to [62], it is uncommon for an error to escape being detected by ECC and be found only by CRC. As such, the configuration controller relies solely on the frame ECC for error detection. The Readback CRC is not used for the reasons that will be presented in Section 4.3.2.

For the SEM operation, the user needs to specify the RBK_FAR and NUM_OF_WORDS at IBUF addresses 0x00E and 0x010 respectively (see Table 4.3). During readback, a copy of the current frame being read is kept in the IBUF. At the end of each fame readback, the FRAME_ECC drives its ECCERROR signal High if an error is found. For a single-bit error, the flipped bit is corrected by the IFSM using the information provided by the FRAME_ECC, and written back to the CMEM.

To avoid the accumulation of the inherent overhead of 1.01 μs per each readback for contiguous frame readback, the readback command can be issued for a chip area of interest and the FRAME_ECC continuously monitored for an indication of error. Immediately an error is indicated, the ongoing readback is aborted in 5 clock cycles, which is only 5% of the 101 that would be incurred if readback was frame by frame. A single-bit error is indicated on the ECCERRORSINGLE port of the FRAME_ECC as a ‘1’. The index of the 32-bit frame word that contains the flipped bit is reported by SYNWORD[6:0] while SYNBIT[4:0] points to the bit position in the word. To save time, rather than flip the erroneous bit inside the IBUF, and as such incur BRAM read and write cycles, the affected bit is flipped when the frame is being written back to the CMEM. In the terminology of the Xilinx’s SEM IP [62], the SEM operation has support for both *Repair* and *Replace* correction methods. The Repair method can correct single-bit errors while the Replace method can correct multi-bit errors but requires access to a

golden bitstream stored externally. Multi-bit errors are detected by checking that ECCERROR is a '1' when ECCERRORSINGLE is a '0' and alerted to the user on the *ecc_mbu_error* port of the IFSM, with the erroneous FAR presented on the *ecc_error_far* port (see Figure 4.2 and Table 4.1). When a single or double-bit error is found the FRAME_ECC asserts the ECCERROR signal. The CRCERROR is used to signal a CRC error and is useful when the Readback CRC is used.

It is important to note that the FRAME_ECC detects errors by checking if the frame being read has changed from the original configured value, its error detection does not cover changeable memory cells like FFs, LUT-RAMs, SRLs, and BRAMs. As such, LUT-based memories are masked using the GLUTMASK_B bit during readback (see the CFRT template in Appendix A.2). BRAM contents are bypassed by the ECC circuit during readback.

4.3.2 Selective-Area Scanning for Soft Error Mitigation

In reliability-centric systems, the ICAP has to be shared between two important system functions, namely, task configuration and SEM. These two functions are indispensable in reliable systems and the problem of multiplexing access to the ICAP between them is thus an important one.

It is obvious that since the ICAP is a single resource, a bottleneck or contention might arise if the sharing of this resource is not well coordinated, especially when real-time requirements are considered. For instance, if the ICAP is occupied by more configuration processes, then reliability can be affected, as the rate at which the chip is scanned reduces. At the same time, too much dedication of the ICAP resource to error mitigation can delay tasks from being configured. In severe cases, this can result in missed task execution deadlines, a situation that is particularly intolerable in real-time systems (e.g., planetary rovers).

The maximum theoretical throughput of the ICAP is 400 MB/s. Based on the recommendation by Xilinx, more than 99% of this bandwidth should be dedicated to SEM [187] for a full device coverage. Since reconfiguration time is limited (< 1% of 400 MB/s) this might necessitate a bigger FPGA than would normally be required if tasks could be swapped in and out more often. This could increase device cost. The

Xilinx SEM IP cannot be pre-empted. However, it is impractical to have a 100% occupation of the FPGA, that is, there are always chip areas unused at any given time. Also, in mixed-criticality systems, the tasks have varying degrees of reliability constraints [127][200]. As such, selecting only the occupied areas for scanning, rather than the entire chip, reduces SEM mitigation time, thereby releasing more time for reconfiguration. Further restrictions can also be applied, where more frequent scanning occurs for those areas on the chip occupied by critical tasks; with relatively non-critical tasks scanned less frequently since the failure of a non-critical task is deemed not detrimental to the system. Any soft errors in areas not currently occupied are repaired when those areas are configured with new tasks.

It should be noted that the recommendation in [187] that SEM coverage time should be > 99% is based on a continuous scanning of the entire chip area. However, instead of scanning the entire FPGA in each SEM cycle, we propose that only the areas where tasks are configured be scanned to reduce the time spent on SEM. This requires tasks to be floor-planned to occupy only as much area as required. Since the internal *Readback CRC* engine of the FPGA cannot be directed to scan a specific area, a custom SEM controller that is able to do so is required. However, the developed configuration controller already has a readback scrubbing functionality. As such, by performing readback scrubbing using a selective-area scanning, it is possible to reduce the amount of time dedicated to device-wide SEM and make more time available for reconfiguration.

The idea is to use the SEM operation to readback-scrub only the regions occupied by tasks by specifying a contiguous number of frames. For multiple contiguous frame readback, the IFSM aborts readback once an error is flagged by the FRAME_ECC. Furthermore, it should be noted that the FRAME_ECC primitive asserts its SYNDROME_VALID signal for one clock cycle at the end of each frame of readback. To ensure a correct abort of readback, the IFSM checks SYNDROME_VALID in conjunction with the ECCERRORSINGLE signal to ensure that a complete frame has been read back before triggering an abort and carrying out further operations. In addition, to simplify the tracking of the location in the IBUF of the erroneous read-back frame, the same frame location in the IBUF (from address 0x073 to address 0x0D7) is

used all the time for buffering no matter how many frames are being checked in a single selective-area SEM operation.

Moreover, the FRAME_ECC can be configured to store on its FAR output, the frame address (stored in a special register called *Error Frame Address Register* (EFAR)) where an error is last found or the frame address that is read last. The former is especially useful when a contiguous number of frames is read back in a single operation, in which case it becomes ambiguous to rely only on the error signals on the FRAME_ECC ports. However, the combination of SYNDROME_VALID and ECCERRORSINGLE are relied on in this work, in which case, it is inconsequential whether the FAR or the EFAR is reported on the FAR output as both are the same when the readback is aborted at the end of the frame in which an error is detected.

4.3.3 Fault Injection Support

To perform a fault injection analysis using the CAM, the RBK and CWR operations are called upon. A target frame is read back and a bit is flipped by accessing the IBUF through the Operation Controller interface. CMEM write (WCFG) commands (see Figure 4.9) are loaded into the last 9 locations of the CWR's frequent command memory space in the IBUF. In conjunction with the already read-back frame stored at IBUF_DATA_ADDR and zeroes in the remaining IBUF locations, this forms a complete frame write operation bitstream consisting of commands, a frame of data, and a pad frame, with the OES sequence loaded by the IFSM to complete the operation.

A subsequent RBK operation is then used to confirm that the bit has been flipped in the CMEM. This check can as well be done by performing an SEM operation, which reads back the target frame and corrects the single-bit error caused by the flipped bit. However, since this is all transparent to the user, the original flipped position can still be checked in the IBUF after the SEM operation completes. A final RBK can be performed to confirm that the bit flip has been corrected.

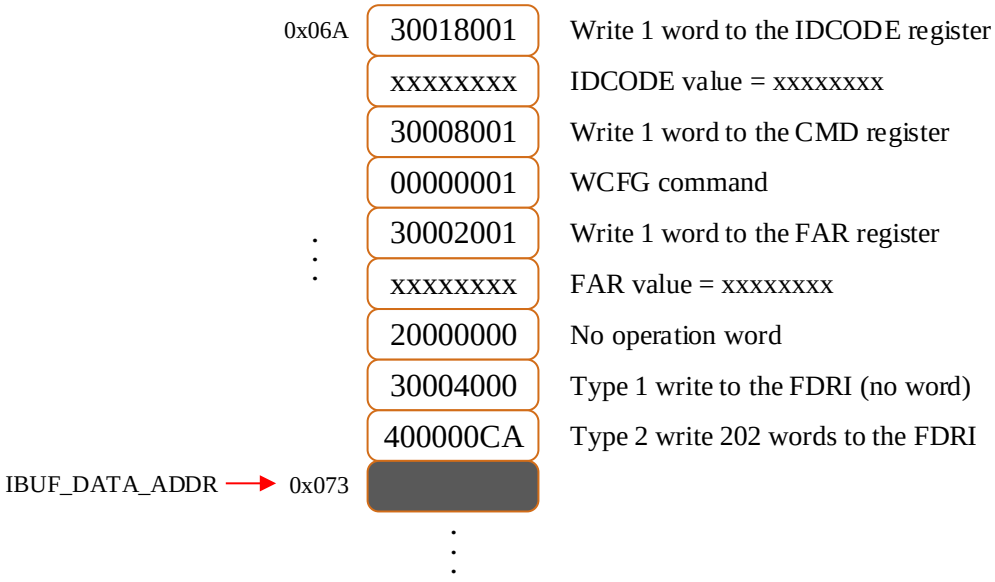


Figure 4.9: CWR frequent commands for fault injection

4.4 Configuration Error Monitoring and Recovery

To improve the availability of the ICAP controller and the host device when deployed in a high-end application, it is important to be able to detect errors during configuration operations. To do this, a diagnosis of the configuration status can be done by monitoring the un-swapped least significant byte D[7:0] of the ICAP's output port. According to [37], the least significant nibble D[3:0] of this byte is supposed to be fixed to "1111". However, this is found to be otherwise; the bits are actually fixed to "1011" for an unknown reason. Apart from this, the other status bits are as expected (see Table 4.7). Bits 7 and 6 are used to monitor configuration error and the status of the ICAP synchronization respectively. While there are other status bits only these two are required to detect configuration errors.

The CFG Controller continuously monitors the configuration error status bit and once an error is detected, the operation is halted; the STAT (status) register of the FPGA is immediately read and presented on the IBUF for the user's attention. Table 4.8 shows the STAT register bits of interest. If the cause of the error is determined to be the CRC, in which case an incorrect pre-computed CRC checksum has been loaded, the user can reset the configuration interface by issuing the RCRC command (0x30008001 followed

by 0x00000007). This can be done with the RGR operation without taking the device offline. However, if there is an IDCODE error the entire device has to be reconfigured or a fallback reconfiguration used [47]. An IDCODE error occurs if there is an attempt to write the FDRI register without a successful device ID check.

Table 4.7: ICAP configuration interface status bits

D[0:7] (*Unswapped*)	**D[7:0] (*Swapped*)**	**Configuration Error Status**	**ICAP Sync Status**
0x9B	0xD9	No error	Not synched
0xDB	0xDB	No error	Synched
0x5B	0xDA	Error	Synched
0x1B	0xD8	Error	Not synched

Table 4.8: Description of the 7 series FPGA's status register

Name	**Bit Index**	**Description**
ID_ERROR	15	Attempt to write to FDRI register without successful DEVICE_ID check. 0: No ID_ERROR, 1: ID_ERROR
CRC_ERROR	0	0: No CRC error, 1: CRC error

It might be worth noting that configuration error detection in the ICAP of the UltraScale architecture is more straightforward. A dedicated Active-High PRERROR port on the ICAP can be monitored to check if there has been an error during partial reconfiguration [201]. Nevertheless, the STAT register still has to be read in order to know the source of error.

4.5 Resource Utilization and Performance Evaluation

The resource utilization of the proposed ICAP controller and by extension, the CAM, is greatly influenced by the coding style used where configuration sequences, templates, and circuitries (in the form of states of the IFSM) are reused as much as possible. This has given rise to a relatively small resource footprint considering the

rich feature set. This has not been at the expense of throughput with the various CMEM read and write operations incurring time overheads within and lower than the range of state-of-the-art controllers.

4.5.1 Resource Utilization Evaluation

Table 4.9 shows the resource utilization of the CAM. The ICAP controller consumes only 234 slices and 1 BRAM36 for all the implemented features and operations. This represents a very small resource footprint even for the smallest of devices, thanks to the efficient coding style employed. For instance, this utilization is only 24.95% on the smallest 7 series device (Spartan-7, XC7S6) and a minute 0.17% on the biggest 7 series FPGA (Virtex-7, XC7VH870T). With this footprint, a TMR implementation to harden the controller against errors would triplicate the utilization to 702 slices, which is still small enough to be accommodated in the smallest of 7 series devices, with enough FPGA real estate available for user designs.

For bitstream data transfer from an external memory to the IBUF, this work has proposed the use of a DMA-capable Data Mover to ensure that the host processing system is not bogged down with data transfers. The free space in the IBUF can also be monitored as the IFSM loads frame data from the IBUF to the CMEM. The Data Mover can thus, transfer bitstreams into the IBUF in close step with the IFSM. The Data Mover is implemented with an AXI Central DMA [202] and the entire Buffer Arbiter consumes a total of 377 slices with 1 BRAM36 used as the IBUF. As such, the entire CAM consumes a total of 611 slices.

Table 4.9: Resource utilization of the CAM in the 7 series FPGA

Resource	ICAP Controller	Buffer Arbiter	CAM
Flip-Flops	330	1,145	1,475
LUTs	654	974	1,628
BRAM36s	0	1	2
Slices	234	377	611

The resource overhead can be further reduced in a ROS in which some of the implemented operations are not required. Nevertheless, the controller is small enough to be used as is without trimming the functions in RTL. In terms of comparison with existing ICAP controllers, it is not a straightforward affair as varied differences between the features used and the devices targeted. Nevertheless, a closely-related work, but one implemented for the Virtex-4 FPGA, is the controller in the Internal Configuration Manger (ICM) developed in [203], where a total of 803 slices are used. A lighter version of this controller is presented in [167], with an utilization of 609 slices. The AC_ICAP [166], which is based on the 7 series family has a fewer set of features, yet consumes 690 slices even with the features trimmed. Table 4.10 shows the comparison of the ICAP controller in CAM to these related controllers and others. An improvement in resource utilization which ranges from about 20% to 71% is observed, noting that most of the compared controllers provide only a subset of the functionalities that CAM's ICAP controller provides.

Table 4.10: Resource overhead comparison of ICAP controllers

Controller	**Device**	**Resource**			
		Slices	***FFs***	***LUTs***	***BRAMs***
ICM [203]	Virtex-4	803	-	-	1
AC_ICAP [166]	7 series	690	1161	1667	7
Xilinx PRC [204]	7 series	-	1203	1170	0
AXI_HWICAP [205]	7 series	291	688	538	0
CAM (This Work)	7 series	234	330	654	1

4.5.2 Throughput Evaluation

For throughput evaluation, the correct functionalities of the operations of the controller are confirmed by using configuration frame data prefetched into the IBUF. This also allows for the characterization of the raw speed performance of the controller. For the various operations of the ICAP controller, Table 4.11 shows the times measured for an operation on a single frame (404 bytes). The ICAP is clocked at its recommended maximum frequency (f_{ICAP_MAX}) of 100 MHz. Measurements are taken by adding a

clock cycle counter to the controller and using the Xilinx *Integrated Logic Analyzer* (ILA) [206] to observe the counter's value.

Table 4.11: Time overheads for the operations of the ICAP controller at 100 MHz

Operation on 1 Frame	**Latency (µs)**		
	Original	***Rolling***	***Optimized***
Readback	2.49	2.38	2.30
Configuration (Basic-CFG)	2.61	2.54	-
Configuration (MFW-based, non-BRAM)	1.71	1.64	-
Configuration (MFW-Based, BRAM)	1.79	1.72	-
Read Modify Write (non-BRAM)	3.83	3.72	3.64
Read Modify Write (BRAM)	3.91	3.80	3.72
Blanking (non-BRAM)	1.56	1.45	1.41
Blanking (BRAM)	1.64	1.53	1.49
Register Read	0.37	0.26	0.22
Custom Write (WCFG+1 frame + 1 pad)	2.55	2.44	2.40
Operation Abort	0.05	0.05	0.05
SEM Scan (no error detected)	2.49	2.38	2.30
SEU Detection and Correction	3.79	3.68	3.56
MBU Detection	2.49	2.38	2.30
MBU-Triggered Frame Replacement	1.71	1.64	-

The "Original" latencies Table 4.11 refer to the exact overhead measured with the ICAP controller used as is. However, because it is possible to leave the ICAP synchronized and allow one operation to "roll" to the other one without ICAP desynchronization, the SYNC words in the OSS sequence and the DESYNC words in the OSS can be trimmed. The overheads in this case are shown in the "Rolling" column in Table 4.11. In addition, the GLUTMASK_B command packets can be bypassed or set to a default value in the full bitstream if the setting will not change in runtime. Therefore, the "Optimized" column refers to the operations carried out with the GLUTMASK command packets trimmed in addition to "Rolling".

When a BRAM frame is loaded using the MFW template, eight NOOP words are written after the MFW command (see Figure A.6 in Appendix A). This accounts for the different timing behaviours reported for BRAM frames for any operation that uses the MFW feature. Considering the basic read and write operations, Basic CFG understandably has the greatest latency of 2.61 µs for a single frame because of the pad frame. Likewise, the RBK operation clocks in at 2.49 µs. A non-BRAM frame writing incurs the lowest configuration time of 1.71µs when the MFW-based CFG is used.

For the BLK operation, a single frame can be blanked in 1.56 µs for a non-BRAM frame and 1.64 µs for a BRAM frame. Compared to using a normal blanking bitstream, the BLK operation saves at least 40% configuration time for a single frame. This figure would increase as the number of frames blanked in a single operation increases.

SEM scan (for both an SEU and an MBU) clocks in at 2.49 µs for a single frame and 1.01 µs for each additional frame in a selective-area operation on a contiguous number of frames. A single-bit upset can be detected and corrected in only 3.79 µs while a frame that has suffered an MBU can be replaced in 1.71 µs per erroneous frame (with the golden frame already fetched into the IBUF). The time for an MBU correction by replacement is heavily influenced by the time needed to access the external memory and transfer data. The time shown here does not consider the frame data movement overhead. Nevertheless, once the golden frame is moved into the IBUF, the time to write it is constant at 1.71 µs.

The latency in bitstream transfer from an external memory to the IBUF depends on the specific data transfer method used. Bitstream transfer takes a latency of 3.993 ms to transfer bitstream data from an external SD card in an example CDMA transfer of 2 kB.

Table 4.12 shows the comparison of the latencies of the basic read and write operations on a single frame. The ICAP controller in this work proves to be comparatively better, providing an improvement of about 30% in configuration latency and about 4% for readback. It is worthy of note that the ICM is targeted at Virtex-4, which has only 41 32-bit words in a frame. As such, the times reported here show that CAM is far more time-efficient considering that a frame in the 7 series is 101 32-bit words.

Table 4.12: Comparison of the basic operation timing behaviours

Controller	Device	Operation Latency (µs)	
		Read Frame	***Write Frame***
This Work	7 series	2.30	1.64
AC_ICAP [166]	7 series	2.39	2.33
ICM [203]	Virtex-4	2.28	1.95

The configuration throughput of the proposed controller is measured by using tasks with various area utilizations and the calculated throughputs averaged out to get a more representative value (see Table 4.13). From this, the average throughput of the proposed ICAP controller is evaluated as 379.70 MB/s, which just short of the theoretical maximum bandwidth (400 MB/s) of the ICAP at 100 MHz. This throughput is only achievable when the IFSM is not stalled at any time waiting for bitstream transfers from the external memory. The buffer free space monitoring proposed is a mechanism that can be exploited to ensure that frame data is made available as soon as possible to avoid stalling the IFSM.

Table 4.13: Configuration throughput evaluation

Task	RM Size			PB Size (kB)	Configuration Time (µs)	Throughput (MB/s)
	CLB	***BRAM***	***DSP***			
Task 1	2	0	0	29	74.32	379.67
Task 2	3	1	0	105	270.36	379.52
Task 3	4	1	1	131	335.00	379.90
Average Throughput						379.70

In order to provide a quick means of evaluating the expected configuration time of a task that is to be configured using the proposed CAM raw throughput, bitstreams are generated for the smallest RMs and FPGA resource block types and these are used to construct templates (equations) for determining partial bitstream size (see Table 4.14). The purpose of this is to support external memory storage budgeting and pre-implementation evaluation of a task's configuration throughput.

Table 4.14: Partial bitstream size evaluation template for the Basic CFG operation

Partition		**Partial Bitstream Size (Bytes)**	
Single	***Block Type***	***N = Num. of Frames***	***N = Block Columns***
	CLB	500 + 404 * *N*	500 + 404 * 36 * *N*
	DSP		500 + 404 * 28 * *N*
	BRAM		500 + 404 * 128 * *N*
Multiple	***Block Pair Type***	***N = Num. of Frames***	***N = Block Pairs***
	CLB-CLB	500 + 404 * *N*	500 + 404 * 72 * *N*
	CLB-DSP		500 + 404 * 64 * *N*
	CLB-BRAM	936 + 404 * (N_{CLB} + N_{BRAM})	936 + 404 * 192 * *N*
Slice	***Slice-Based Partition, N = Number of Block Pairs***		
	936 + 404 * (72 * $N_{CLB\text{-}CLB\_PAIRS}$ + 64 * $N_{CLB\text{-}DSP\_PAIRS}$ + 192 * $N_{CLB\text{-}BRAM\_PAIRS}$)		

These equations can be used to evaluate any task whose area occupation is known without actually generating bitstreams or using the controller. This would prove invaluable for a rapid system design and offline evaluation. Given resource utilization breakdown of a task in terms of CLBs, BRAMs, and DSPs, it becomes quite straightforward to estimate the bitstream size overhead and configuration timing behaviour of the controller for the task. However, a true reflection of the bitstream size would be that calculated from the utilized slice reported by Vivado as there is usually no 100% utilization of the CLBs in an RP. Thus, Table 4.14 also provides an equation for a slice-based partitioning.

The task bitstreams used for constructing the equations follow the format presented in Figure A.5 of Appendix A. This format does not include setup commands like GLUTMASK_B setting. If a task in question has to use certain setup commands the number of words (converted to bytes) before the occurrence of 0x30002001 can be added to the bitstream size. Note that the SYNC words and the WCFG command packets should not be included.

In addition, it is possible to construct equations for the latencies of relevant operations of the controller as shown in Table 4.15. Therefore, given the resource block type and the number of frames (*N*), the latencies for a given task can be easily computed

and in conjunction with Table 4.14, the operation throughputs can be also be determined. For the Basic CFG operations in Table 4.15, the pad frame is not counted as part of the number of frames and note that the RMW can only relocate one frame at a time to multiple frame address locations (*L*). *N* is the number of frames to access in a single operation.

Table 4.15: Latency templates for selected operations of the ICAP controller

Operation	**Time for 1 Frame (µs)**	**Latency for *N* Frames & L locations (µs)**
Readback	2.49	1.48 + 1.01*N*
Configuration (Basic-CFG, 1 block type)	2.61	1.60 + 1.01*N*
Configuration (Basic CFG, 2 block types)	4.73	3.72 + 1.01*N*
Read Modify Write (non-BRAM)	3.83	(3.72 + 0.11*L*)**N*
Read Modify Write (BRAM)	3.91	(3.72 + 0.19*L*)**N*
Blanking (non-BRAM)	1.56	1.47 + 0.09*N*
Blanking (BRAM)	1.64	1.47 + 0.17*N*
SEM Scan	2.49	1.48 + 1.01*N*

4.5.3 A Case-Study Application

To evaluate the ICAP controller with a practical application and illustrate the advantage of the proposed selective-area scanning, we draw a case study from the NASA JPL's *Compositional InfraRed Imaging Spectrometer* (CIRIS) [207] implemented on a Zynq-7000 (XC7Z100) FPGA. The CIRIS is one of the new-generation NASA instruments proposed to search for life indicators in Jupiter's moon, Europa [208]. The total resource usage of the CIRIS data processing task is 7,781 slices, 164 DSP48s, and 176.5 BRAM36s [209] (see Table 4.16). A floorplan on the XC7Z100 chip that would accommodate all the resources (especially the BRAM36s) requires a selection of a larger region than required by the slices and DSP48s alone. This translates to area occupations of 20.19% for slices, 23.76% for DSP48s, and 26.49% for BRAM36s. This is equivalent to 48 CLB-CLB pairs, 24 CLB-DSP pairs, and 20 CLB-BRAM pairs.

Table 4.16: Resource utilization of the CIRIS data processing circuit

Component	**Slices**	**DSP48s**	**BRAM36s**
CIRIS Data Processing Circuit [209]	7,781	164	176.5
Selectable Resources	14,000	480	200
Available in the XC7Z100 chip	69,350	2,020	755
Percentage Resource Used	11.22%	8.12%	23.38%
Percentage Area Occupied	20.19%	23.76%	26.49%
Configuration Frames	5,040	672	2,560
Total Configuration Frames (from Block Pairs)	8,832 frames		
Partial Bitstream Size	3,485.41 kB		

By using Table 2.11 we estimate that a total of 8,832 frames are needed for configuring or reading back the task. This gives a PB size of 3.4 MB and a Basic-CFG configuration time of 8.9240 ms, with a throughput of 381.42 MB/s. A selective-area SEM operation would scan these frames in 8.9218 ms while the Xilinx SEM IP [62] would still incur the full device scan time of 34.3 ms. This is a saving of 25.38 ms (74%) of ICAP access time and this can be allocated to reconfiguration (see Table 4.17).

Table 4.17: Configuration and scan times for the CIRIS task

Operation	**Operation Latency (ms)**
Configuration	8.92404
Readback	8.92180
Blanking	0.95222
Scan Time (the SEM operation in this work)	8.92
SEM Scan (Xilinx SEM IP) [62]	34.30
Time Saved for Reconfiguration	25.38
Percentage Time Saved	**74%**

It should be noted that the more occupied the FPGA is, the lower would be the time saved, and indeed, there are other circuits that may be critical in the spectrometer example used [210]. However, if the FPGA is fully occupied, which is an unlikely situation, advantage can be taken of the different criticalities of the tasks to scan the

more critical ones more often and the less-critical ones less. This is possible because our ICAP Controller's SEM engine allows selective-area scanning. The time saved can be put towards task reconfiguration. Since there is more time for reconfiguration, more tasks can share the FPGA in time and space, and the size requirement for the chip would be lower leading to cost savings while not compromising reliability.

4.6 Chapter Summary

A high-performance and reliable configuration controller is an indispensable component of reconfigurable systems, especially those that are based on FPGAs. This chapter has presented a configuration controller that implements functionalities that are crucial for task loading and deloading, and several reliability-enhancing functionalities (e.g., SEU and MBU mitigation and internal register reading for device diagnosis) for reconfigurable computing. Without loss of generality, the Xilinx 7 series FPGA family has been used as a target for the proposed controller. Resource utilization and throughput evaluations have revealed that this controller outperforms similar state-of-the-art controllers saving up to 71% area overhead and having 30% less configuration latency for a single frame, with an average throughput of 379.70 MB/s for contiguous number of frames. Benchmarking templates are also provided, targeting an easy evaluation of hardware tasks' bitstream size and configuration overhead with the aim of easing system design when the proposed configuration controller is to be deployed.

While reliability and availability have been some of the key focus, the proposed controller is generic enough to find application in a wide variety of scenarios. The reliability of the controller itself can be easily improved by adopting a TMR implementation. This does not have a serious impact on resource footprint because the controller already has a very small footprint considering the implemented feature set and the state-of-the-art.

Chapter 5

5

Secure and Efficient Hardware Task Relocation Framework for Reconfigurable Computing

One key RC service, which has not been given prime attention is security [24] even though research works already appeared almost two decades ago highlighting the need for circuit protection [20]. Meanwhile, the fact that FPGAs are now being used in high-end applications has served to increase interests in FPGAs and the value of IP cores that run on them, thanks to the adoption of FPGAs by tech giants like Microsoft [39] and Baidu [40]. IP core vendors pay heavily in monetary terms and development time to design these IP cores. As a result, the protection of this investment is of paramount importance. However, malicious attacks on FPGAs have aimed to exploit the security shortcomings of FPGAs to allow attackers to steal these IPs or cause undesirable effects.

As a deterrent to these attacks, major FPGA manufacturers (like Xilinx) have introduced secure bitstreams, which are bitstreams protected with both authentication and encryption (see Section 2.3). However, as FPGA programming technology and techniques evolve, renewed thoughts have to be given to the impact of encryption on the format of bitstreams and how this affects emerging hardware task management techniques. This is the case with PBR, which is often deployed for permanent damage mitigation. As existing PBR methods are not amenable to encrypted PBR (see Section 3.2.4), excessive resource and time overheads can result in a bid to ensure the continuous protection of IP cores in the face of PBR. This chapter delves further into these issues and presents a novel resource- and time-efficient PBR mechanism to address the situation.

The techniques and mechanisms reported in this chapter have been included as part of the author's publications [211] and [212]:

- **A. Adetomi**, G. Enemali, and T. Arslan, 'Relocating Encrypted Partial Bitstreams by Advance Task Address Loading', in *25th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2017)*, 2017, pp. 188–191.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Towards an Efficient Intellectual Property Protection in Dynamically Reconfigurable FPGAs', in *2017 Seventh International Conference on Emerging Security Technologies (EST)*, 2017, pp. 150–156.

5.1 The Challenges with Encrypted PBR

With respect to partial bitstream relocation, there are a few challenges with the Xilinx's secure bitstream format. One limitation with the use of encrypted partial bitstreams in the 7 series FPGAs lies in the restriction on the configuration interfaces. The readback of the configuration memory through the JTAG and SelectMAP interfaces is not possible when an encrypted bitstream is loaded into the device. While this restriction does not apply to the ICAP interface; nevertheless, the configuration bandwidth is reduced by a factor of four to a theoretical value of 100 MB/s as the ICAP interface only accepts encrypted bitstreams through its 8-bit input bus [47]. This increases the configuration overhead during relocation and is a price that must be paid to ensure bitstream security in the 7 series. Indeed, Xilinx has addressed this in the UltraScale architecture, which ships with an ICAP that only supports a 32-bit port width for both unencrypted and encrypted bitstreams [201].

Moreover, the state-of-the-art vendor secure bitstream format introduces a bottleneck for PBR. PBR requires the runtime manipulation of the bitstream's frame addresses before it is delivered to the configuration interface. As such, access to the readable plaintext decrypted bitstream is needed. However, the frame addresses are specified in the encrypted portion of the secure bitstream and are as such, in cyphertext format. Meanwhile, to prevent a breach of security during reconfiguration, the on-chip decryptor in the FPGA feeds the plain decrypted data directly to the configuration interface; and as such, access to the decrypted data is denied. To overcome this, a

custom dedicated AES decryptor circuit can be used. However, this circuit usually takes a considerable amount of FPGA resources [213] [214] and time (see Sections 5.6 and 5.7).

To address this issue in a resource-efficient and time-saving manner, a secure bitstream format that is friendly to relocation is required. Moreover, there is usually more than one instance of frame address loading. This means a user cannot simply state a FAR address before loading the encrypted frame data, we have to split the bitstream into a number of parts corresponding to the number of frame addresses in the bitstream. For each section, the FAR is loaded unencrypted before the encrypted configuration data. This calls for a unique software that can accept an unencrypted bitstream, remove the FAR loading commands, split the bitstream, and perform other necessary bitstream manipulations; and a unique configuration controller that can generate the FAR address in runtime and load the bitstream parts after loading the corresponding generated FAR addresses.

5.2 Relocation-Aware Secure Bitstream Format

An experimental study of the reconfiguration of Xilinx FPGAs reveals that the frame address which points to the location to be configured in the FPGA can be written to the FAR in a separate command sequence in advance of the frame data loading itself. Having observed that the FPGA remembers the content written to the FAR, the proposed solution to expensive encrypted PBR involves using a software algorithm to remove the FAR loading command and the FAR address itself from the bitstream before it is encrypted and issuing this unencrypted command inside the FPGA in runtime through the ICAP configuration port.

To understand why this is possible, it is fitting to note that the FPGA would normally wipe off the configuration memory when an unencrypted data loading is initiated subsequent to the loading of an encrypted bitstream. However, because the ICAP is a trusted interface, it is allowed to read back data and also write unencrypted data even when encryption is used. Meanwhile, in order not to compromise security,

the ICAP signals are not routed out of the FPGA [47] and the user is prevented from issuing readback commands when encryption is used.

Since the proposed approach to encrypted PBR involves loading the task's plain unencrypted frame address in advance of the encrypted frame data, this scheme can be appropriately named *Advance Task Address loading* (ATAL). The unique secure bitstream format used proposed for ATAL is depicted in Figure 5.1. In contrast with the Xilinx secure bitstream format, which has the frame address inside the encrypted body, the ATAL format does not contain any frame address information. The frame address is generated on-chip when needed. The bitstream is reformatted into a global preamble and a number of body parts depending on the number of frame addresses in the original bitstream. Each body part contains a local preamble, a local body, and local postamble. Like in the Xilinx secure format, only the body and the postamble are authenticated and encrypted. The following subsections provide more details about the different sections of an ATAL-formatted bitstream. Note that while an encrypted bitstream is loaded through the ICAP one byte per clock cycle, references to a "word" of data in the bitstream still retain the original meaning of 32 bits.

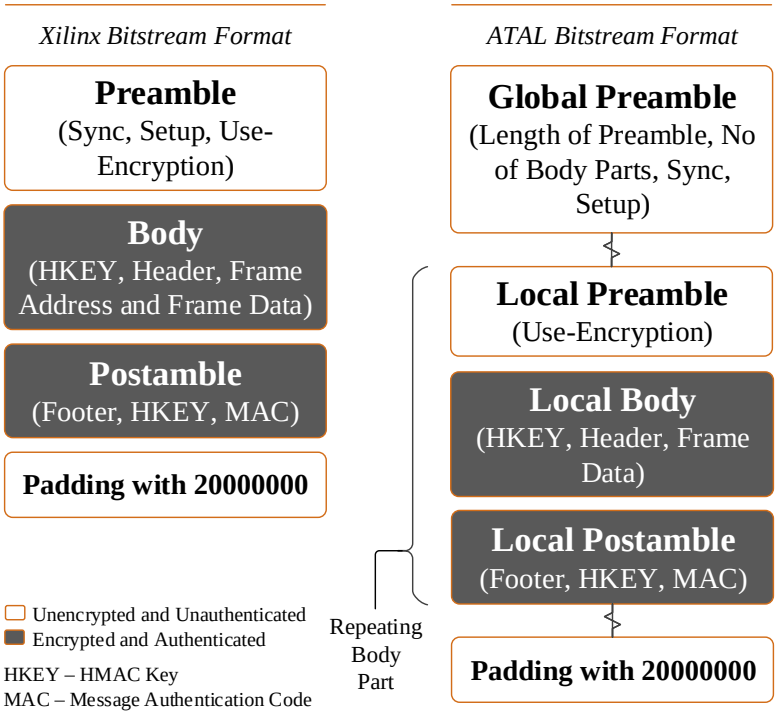


Figure 5.1: Secure bitstream formats from Xilinx and for ATAL

5.2.1 Global Preamble

The global preamble is similar to the preamble in the Xilinx bitstream format as it is composed of bus width auto detection words (0x000000BB followed by 0x11220044) and the synchronization word (0xAA995566). For the 8-bit interface enforced by the use of encryption, the internal configuration bus width auto detection logic would properly align by finding 0xBB followed by 0x11 on bits [7:0] of the ICAP's output.

Apart from the above, the ATAL's global preamble also has prepended, the number of bytes in the global preamble (N_BYTS_GPMBL) and the number of body parts (N_BDY_PRTS) in the entire bitstream, both concatenated in the same 32-bit word. Bits [31:24] are used to keep N_BYTS_GPMBL, allowing for up to 256 bytes (64 words) in the preamble, while bits [23:0] are used to store N_BDY_PRTS, allowing for up to 16,777,216 body parts. As many as 24 bits are reserved for N_BDY_PRTS to cater for the many body parts expected when a compressed bitstream is parsed by ATAL. For instance, the largest documented 7 series FPGA (7VH870T) (at the time of writing) has a configuration bitstream length of 294,006,336 bits [47]. This is approximately 90,968 frames assuming the entire content of the bitstream is frame data. The worst-case compression scenario would produce a compressed bitstream with 90,968 frame addresses and thus, body parts. This is well below the maximum N_BDY_PRTS in ATAL.

5.2.2 Local Preamble

Since in the un-split Xilinx-formatted encrypted partial bitstream (Figure 5.1), the entire body of commands and data share the same preamble, which includes some FPGA setup commands, the instruction to enable the AES decryptor and load the encryption key from the eFUSE or the BBRAM, the AES IV loading, and the DWC loading, each of the body parts must also have a similar local preamble for proper synchronization with the decryption circuit in the internal configuration logic of the FPGA. Four words are used for the use-encryption command, five words for loading the IV, and two words for the DWC packet. These additional words are added to the beginning of each body part as a part of a local unencrypted preamble.

Figure 5.2 shows the composition of the local preamble. Bit 6 of the CTL0 register has to be set to '1' to instruct the FPGA that the internal AES decryptor engine should be enabled. Also, a selection of the key storage option (EFUSE_KEY = '0' for BBRAM or '1' for eFUSE) at bit position 31 of CTL0 has to be made. All the variable fields are automatically filled in by ATAL's software algorithm (see Section 5.3)

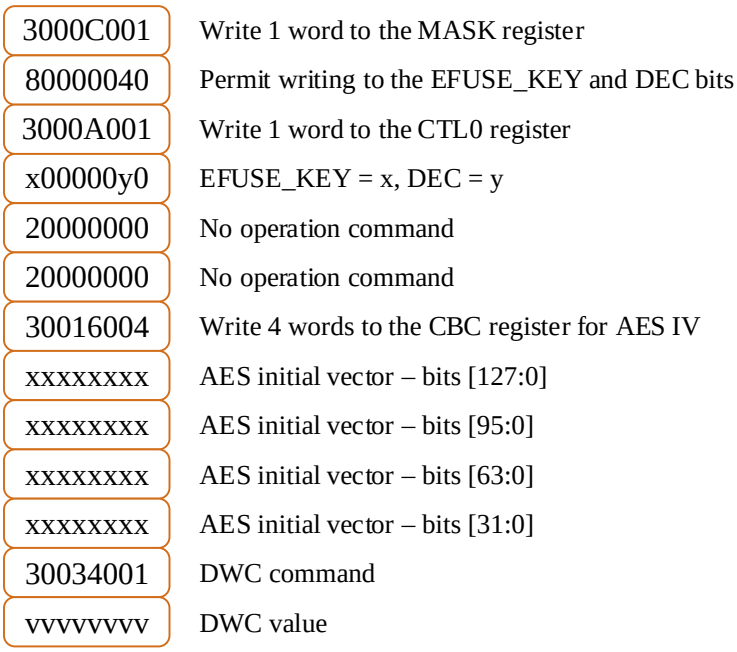


Figure 5.2: Composition of the local preamble of an ATAL-formatted bitstream

5.2.3 Local Body

The local body contains the encrypted configuration commands and frame data. When compared with the Xilinx secure bitstream format, the key difference is that there are no frame addresses in the local body of ATAL-formatted bitstream. The frame address loading commands and the respective FAR values are removed before authentication and encryption and auto-generated on demand in runtime.

Moreover, a careful examination of the unencrypted and encrypted bitstreams generated with Vivado reveals that the FPGA setup commands like resetting the CRC and loading the device IDCODE in the preamble of the unencrypted version are no longer included in the unencrypted preamble of the encrypted bitstream. Since these commands are still required, they must be included before encryption. In ATAL, each

bitstream body part essentially becomes a standalone encrypted PB without device sync and setup commands. As such, ATAL's CRC coverage is at the body part level. The RCRC command is added to the header of each local body before encryption. In addition, since every write to the FDRI requires a successful device ID check, the IDCODE command is included in the header of each local body of every body part. Other user-determined FPGA setup commands retrieved from the original bitstream are added to the header as well.

Noting that one of the aims of encryption is to hide the content of the bitstream from the prying eyes of attackers, it would be good to reveal only the information necessary to inform the configuration interface that encryption is being used. This is what Xilinx has done in their secure bitstream format and the same tradition is kept to in this work. As a result, all the commands in the preamble retrieved from the unencrypted bitstreams are also encrypted along with the first body part. This leaves the local headers and the new preamble added for instructing the FPGA to use decryption as the only unencrypted parts of the processed bitstream.

5.2.4 Local Postamble

The local postamble is composed of footer commands, with the HKEY and MAC appended. The footer holds the command to load the CRC register with the recalculated CRC value. There is one special FAR loading in the postamble of both plain and encrypted Xilinx-formatted bitstreams. The value of the frame address loaded is 0x03BE0000. This represents a column address of 0, top-half row address of 31, and a block type of '111', which does not correspond to any known or documented resource type in the FPGA. The choice of extreme values for the FAR fields suggest that 0x03BE0000 is meant for some form of internal synchronization. It would not be surprising to have no Xilinx FPGA with up to 32 rows in the top half of the device. Keeping in line with this, the ATAL-formatted encrypted bitstream includes in the local postamble of the last body part the packet 0x3000200103BE0000 just before the RCRC command.

5.3 Software Interface for Bitstream Reformatting

The FPGA vendor's bitstream generation tool would not accept an existing bitstream as an input to the synthesis and implementation flow and even if it did, one would not be able to direct it to reformat the bitstream into the unique relocation-friendly ATAL format being proposed. Therefore, a special offline-executed algorithm is needed to perform the operation of reformatting the bitstream. We have named this algorithm *Splixbit* as an abbreviation of "splitting a bitstream" into parts.

Figure 5.3 summarizes the execution flow of the Splixbit software in converting a partial bitstream to an ATAL-formatted one. After loading the bitstream, the tool first processes the header to confirm that the file loaded is a partial bitstream; an error is thrown if that is not the case. After the header is extracted, the bitstream is split into a preamble (sync and setup commands), a number of body parts depending on the number of frame addresses, and a postamble (DESYNC and CRC check commands). The formatting of the bitstream parts includes removing the FAR command and frame address, removing the DESYNC command packet, recalculating the CRC value. Other necessary steps like the addition of IDCODE command are carried out. Next, the HMAC-SHA-256 hashing and AES-CBC-256 encryption are applied to the separate parts, after which the encrypted bitstreams are combined into a single bitstream.

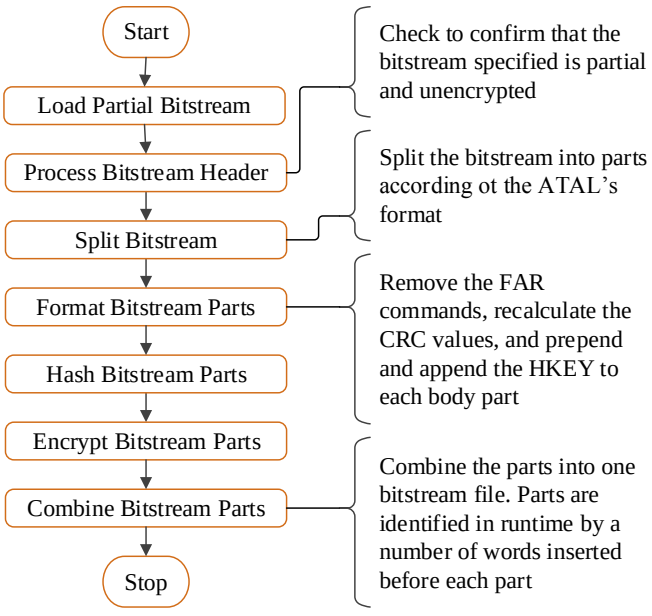


Figure 5.3: Splixbit's algorithm's flow for advance task address loading

The following subsections provide more details and related technical considerations for the algorithm's software interface:

5.3.1 Splixbit File Input

For data input to the Splixbit software, the user can supply both encrypted and unencrypted partial bitstreams. In the case of encrypted bitstreams, the software tool first decrypts the bitstream before performing the necessary splitting, encryption and formatting. Being able to supply encrypted bitstream to the software may be useful in situations where there is no access to the original design in order to generate the unencrypted bitstream. In this case however, the user has to present the tool with the original encryption key file generated by the implementation tool. It is expected that the user should have this since it is needed to load the AES key into the FPGA.

There are different bitstream file format options available in the Vivado. These options, accessible via *Bitstream Settings* in Vivado, can be used to direct the tool to generate some other useful files apart from the BIT file, which is the default binary bitstream file used for configuration through the *Hardware Manager* of Vivado. Some of these files are the raw bit (RBT) file, which contains the same information as the BIT file but in ASCII format, the mask (MSK or MSD) file, useful for verifying readback data, the readback (RBB or RBD) file, useful for bitstream verification; and the binary (BIN) file, which is similar to the bit file but without the header information. Since the header information is not actually uploaded to the FPGA but is used by other Xilinx tools, ATAL could do with the BIN file.

However, the Splixbit algorithm requires the knowledge of the device type in order to prepare the encryption key file (NKY file) correctly. The device type is contained in the header; as such, it is more convenient to retrieve it if the user supplies a BIT file. In order to understand the information in the header and retrieve the device type, we have examined the contents of both unencrypted and encrypted, full and partial bitstreams of 7 series FPGAs. Our findings are presented in Table 5.1. When the bitstreams are viewed using a HEX to ASCII editor, distinct sections, mostly delimited by semicolons and alphabets from 'a' to 'e', are noticed in the header before the configuration interface synchronization words.

To retrieve the target device type, a regular expression (*regex*) is used to search the ASCII-representation of the BIT file. The regex used depends on the type of file whether encrypted or unencrypted. The usage of regex in this work is pretty standard and based on the `Regex` class in the .NET Framework. It should be noted that full bitstreams are not manipulated by the software tool, they are included in Table 5.1 only for comparison and general understanding.

Table 5.1: Bitstream header information (bitstream generated in Vivado 2015.2)

Field Bytes	**Bitstream Type & Text Fields**			
	Full & Unencrypted	***Full & Encrypted***	***Partial & Unencrypted***	***Partial & Encrypted***
16	File Declaration and Header			
Variable	Design Top Module Name			
1	Delimiter			
11	[a]NA	ENCRYPT =YES	[a]NA	ENCRYPT =YES
1	[a]NA	Delimiter	[a]NA	Delimiter
13	[a]NA	[a]NA	BLANKING=TRUE	[a]NA
1	[a]NA	[a]NA	Delimiter	[a]NA
17	User ID			
1	Delimiter			
12	[a]NA	[a]NA	PARTIAL= TRUE	PARTIAL= TRUE
1	[a]NA	[a]NA	Delimiter	Delimiter
61	Tool Version, Target Device, Date, and Time			

[a]NA – Not Applicable

5.3.2 CRC Recalculation

If the user enables CRC check, the Vivado calculates the CRC value of the entire bitstream command and data and appends it to the bitstream (in the postamble of an unencrypted bitstream). Any modification to the bitstream while it is in transit to the configuration interface leads to a CRC error. Because ATAL involves a deliberate modification of the bitstream, the precomputed CRC is effectively rendered void. A simple solution would be to disable the CRC functionality in Vivado. However, this

would be counterintuitive to bitstream integrity. Therefore, to preserve this functionality, the CRC is calculated on each of the body parts using the software and the RCRC and load CRC (0x30000001 followed by the 32-bit CRC checksum) commands are inserted at the beginning and end respectively of each of the body parts.

The CRC is recalculated using the CRC-32 polynomial: $x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^{9} + x^{8} + x^{6} + 1$ [59], which is the CRC-32C (Castagnoli) polynomial 0x1EDC6F41 (normal) or 0x82F63B78 (reversed). The .NET Framework does not come with a built-in CRC checksum calculation library. As such, a custom code implementation is used.

5.3.3 HMAC-SHA Authentication and AES-CBC Encryption

The HMAC-SHA-256 hashing and AES-CBC-256 encryption are applied to the separate body parts, after which the encrypted bitstreams are combined into a single bitstream file. For the HMAC authentication and AES encryption the HMACSHA256 and RijndaelManaged classes of the .NET Framework are used. Padding words are used to ensure that data lengths are a multiple of 128 bits for encryption and 512 bits for authentication, as required for simplifying the MAC computation [43].

5.3.4 Splixbit Graphical User Interface Description

Figure 5.4 shows a screenshot of the Splixbit software interface. The entire software was written in C# as a *Windows Form Application*. There is a main textbox for displaying the unencrypted bitstream loaded into memory. Three text input boxes are provided for the AES key, AES IV, and the HMAC key. In addition, there is an Options box; the user can choose to allow the software to generate the keys and IV, supply them as inputs through the text boxes, or specify a key file (NKY file). Clicking the Load Bit File button opens a file dialog window for the user to specify the bitstream file. After this, the Execute button can be pressed to process the bitstream. A few error control dialogs have also been coded in to aid usage. All the operations carried out are captured in the log output at the bottom of the interface.

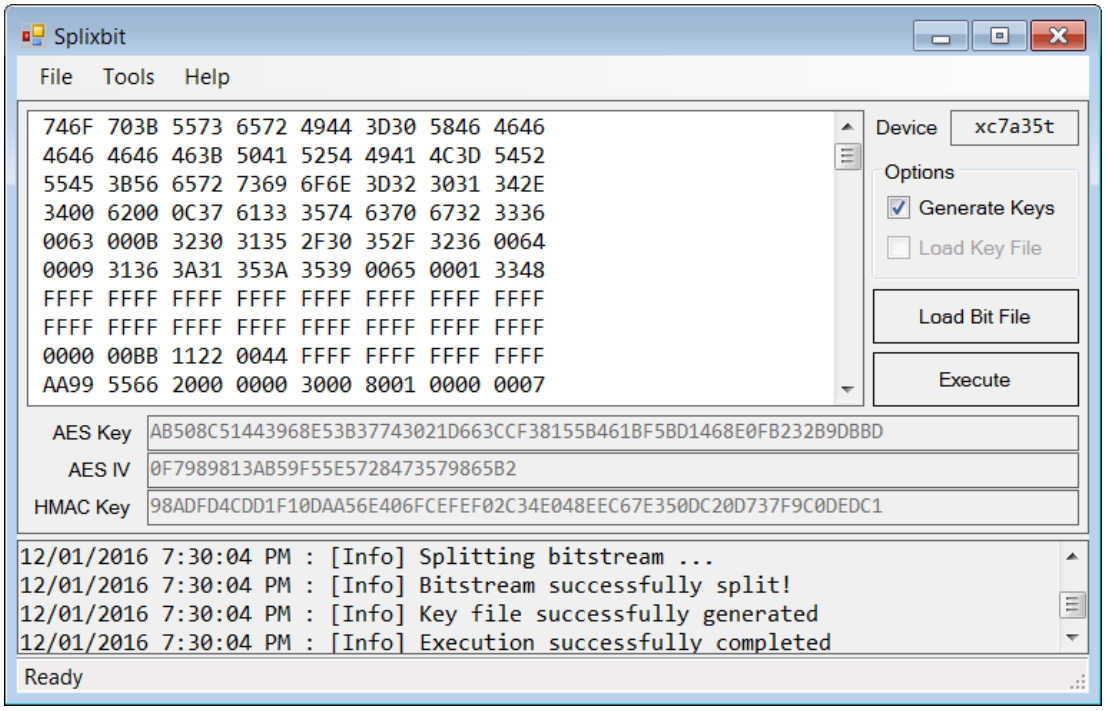


Figure 5.4: Screenshot of the Splixbit software interface

5.4 Hardware Support for Encrypted PBR

In order to deliver the Splixbit-formatted bitstream to the internal configuration interface of the FPGA in run time, a unique configuration flow is required. Existing ICAP controllers cannot be used to directly load an ATAL-formatted bitstream because of the special requirement of specifying plain frame addresses at different points in the bitstream loading. It is pertinent however, to mention that an existing controller that is able to load a Xilinx-formatted encrypted PB would require only a few modifications to get it up and running as far as ATAL is concerned.

5.4.1 Configuration Controller

To control the bitstream loading, a unique ICAP controller (Splixbit hardware) is required. The Splixbit hardware is a minimalistic version of the ICAP controller in Chapter 4, implemented to demonstrate the feasibility of ATAL. There are inevitable similarities between the two. For instance, bitswapping is still required at the input and output of the ICAP. However, a main difference is that the ICAP input for a secure bitstream is restricted to an 8-bit width. This necessitates that the initial 32-bit word that

could be loaded in one clock cycle is now loaded in 4 clock cycles, with the most significant byte loaded first. The effect is the quadrupling of reconfiguration time. The scaled-down version of the ICAP controller is shown in Figure 5.5.

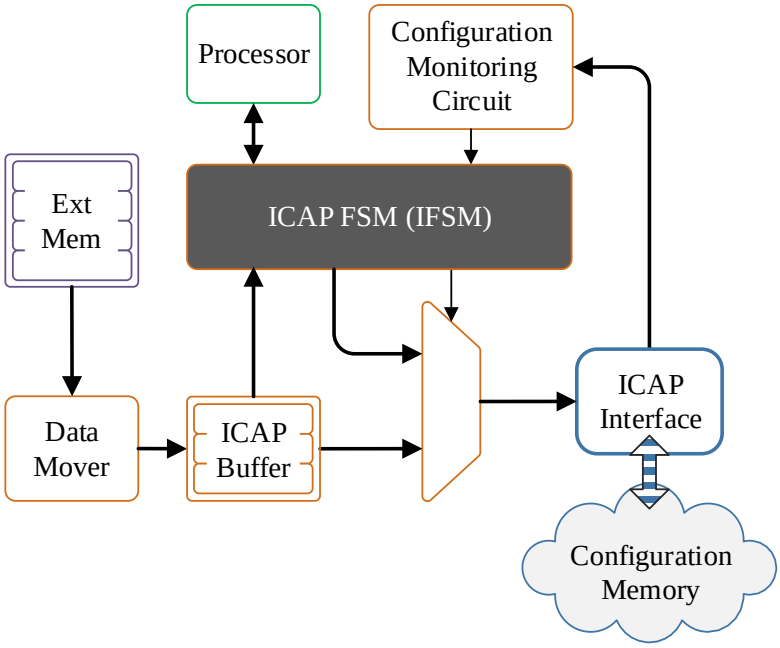


Figure 5.5: Configuration controller for loading Splixbit-formatted bitstreams

The BRAM-based IBUF is implemented as a two-port memory with one port connected to the Data Mover and configured to have a 32-bit data width. This allows the bitstream to be transferred at a much faster rate than the IFSM loads it. The other port of the IBUF is configured for 8-bit data access and connected to the IFSM. This arrangement forces byte loading to take an order where the four bytes in each word are read from the high byte address to the low byte address and the *ibuf_addr_pntr* advanced by 7 to fetch the next word.

For PBR, the FAR command detection and FAR modification circuits are not included. Since the FAR command is no longer in the bitstream, the user is expected to provide frame addresses for the bitstream body parts at appropriate points in the operation. Moreover, the IBUF is simply used for bitstream buffering as no operation sequences and templates are stored in it. The only sequence needed is the first three words (0xFFFFFFFFAA99556620000000) of the OSS and this is simply added (as a

VHDL constants) in RTL. The OSS is needed to resynchronize the configuration interface after a configuration error detection.

Moreover, from the standpoint of configuration error monitoring, in addition to CRC and IDCODE errors, a further factor can be a source of error. The internal configuration logic throws a DEC_ERROR if there is an attempt to write to the FDRI register before or after a decrypt operation. A '1' at bit position 16 of the STAT register indicates a DEC_ERROR while a '0' indicates otherwise.

In terms of the operations of the IFSM, the only operations retained in the Splixbit hardware are configuration for PBR support and the ABT operation for aborting configuration. However, the opcodes are no longer used for selecting operations; the default operation when the *icntrlr_en* is asserted is the CFG operation and an abort can be triggered by setting the *abort* port to '1'. Figure 5.6 shows the control interface of the Splixbit hardware's IFSM and highlights the ports not found on the previous IFSM of Figure 4.2. The *icntrlr_err_src* is a 3-bit port used to alert the user to the three sources of a configuration error. CRC error, IDCODE error and DEC error are reported respectively on bits 0, 1, and 2.

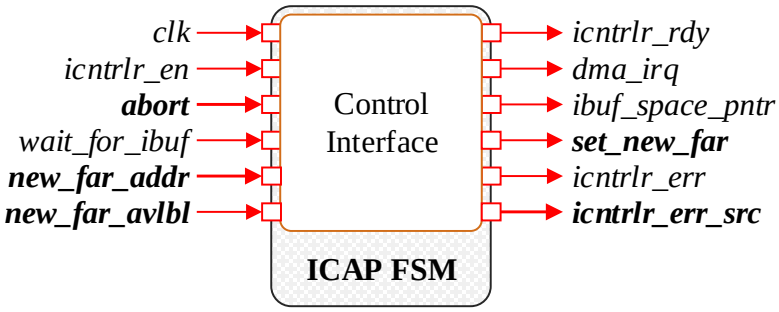


Figure 5.6: Key ports of the Splixbit hardware's finite state machine

5.4.2 Configuration Flow

It is necessary to organize the ATAL-formatted bitstream loading in the order of global preamble, FAR addresses, and body parts as described by the flowchart. As such, the operation of the Splixbit hardware's IFSM follows the flowchart of Figure 5.7. Once enabled, the IFSM first retrieves N_BYTS_GPMBL and N_BDY_PRTS from the first word in the global preamble (addresses 0 to 3), and then loads the global preamble

(Sync and Setup) commands to the ICAP. Appendix C presents example waveforms of the Splixbit hardware's IFSM parsing the global preamble of an ATAL-formatted bitstream.

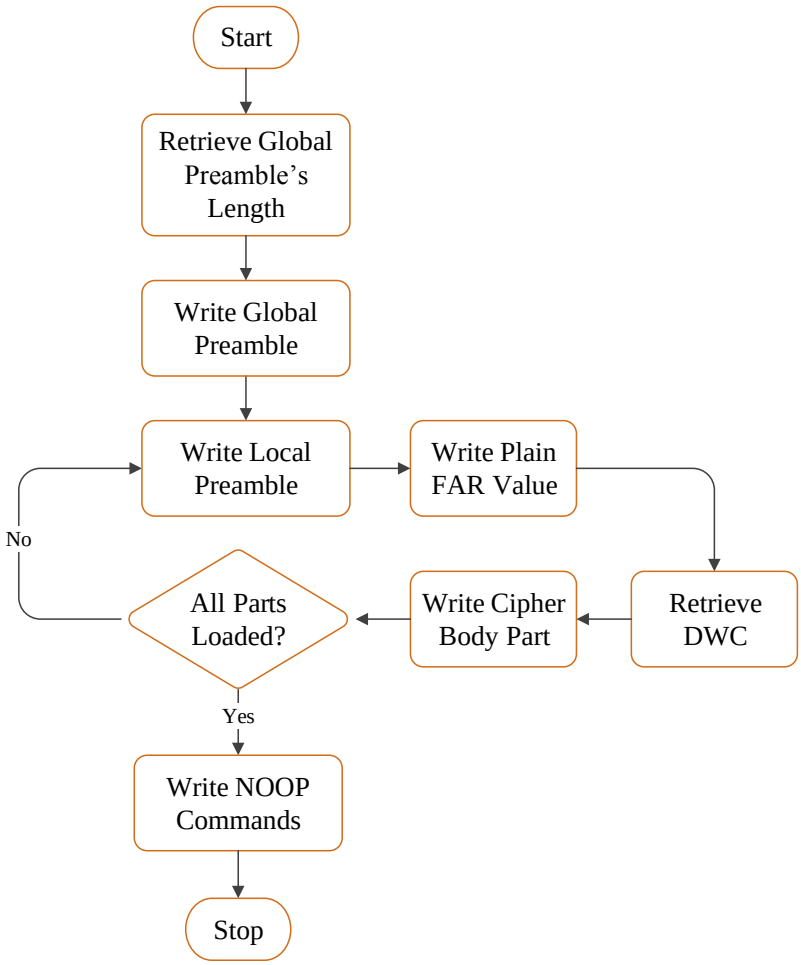


Figure 5.7: Splixbit configuration flow for encrypted bitstreams

The IFSM then loads a new unencrypted frame address supplied through the processor before loading its corresponding local preamble and encrypted frame data. To ensure that configuration does not stall on waiting for an address, the first frame address is supplied before configuration starts, and subsequent frame addresses immediately after a previous one is loaded. Proper handshaking between the processor and the IFSM ensures this. The ports *set_new_far*, *new_far_addr* and *new_far_avlbl* are used for this handshaking. The IFSM asserts *set_new_far* when a new FAR loading is anticipated, the user should provide the new frame address on the input port *new_far_addr* and assert *new_far_avlbl*. The IFSM then picks the new frame address,

deasserts *set_new_far*. The user should deassert *new_far_avlbl* as soon as *set_new_far* goes low.

Between the frame address and the encrypted frame data loading, the local preamble is processed to retrieve (from the DWC) the length of the encrypted data to load, and to write the Use-Encryption commands. Both the local body and the local postamble are all part of the same encrypted block of data and are loaded in a single operation. The cycle of address loading, local preamble processing, and local body and postamble loading continues until all the body parts have been loaded.

5.4.3 Loading Termination

The fact that we are dealing with encrypted bitstreams calls for certain considerations. Since a state machine (IFSM) controls the sending of data to the ICAP, it is important for it to know when to stop loading data. The *End of Startup* (EOS), an *Active-High* signal on the STARTUP primitive of the FPGA would have been the perfect signal to monitor. This signal goes high at the end of start-up after configuration, indicating that the FPGA is ready for operation [47]. However, our experiments with this primitive revealed that the EOS signal remains asserted after the configuration of the full bitstream and does not toggle during dynamic partial reconfiguration. In [37], Xilinx recommends monitoring the configuration bitstream for the DESYNC word (0x0000000D) that signals to the configuration interface that data has been completely delivered. In our case, this word is already encrypted; as a result, we monitor the number of words configured to know when configuration has finished.

It would be a lot convenient to have a signal like the EOS on the ICAP primitive of the 7 series FPGAs. It is worthy of note however, that Xilinx has introduced this mechanism in the UltraScale architecture – there is on the ICAP interface, a PRDONE signal that goes High on PR completion [37]. Unlike in an unencrypted bitstream, the FAR command header 0x30002001 issued to load the frame address is not visible to the IFSM. As a result, the IFSM is in the dark as to when to insert a new frame address for PBR. The IFSM needs to know when the preamble has been loaded so as to load the FAR before loading each of the body parts. Moreover, the IFSM needs to know when the last body part has been loaded in order to stop the insertion of frame addresses.

By looking for the loading of the DWC command header (0x30034001), the IFSM retrieves the number of words to configure for the body part instance and uses the number of body parts retrieved from the global preamble to know when to terminate bitstream loading.

5.4.4 Resource Utilization and Latency of the Splixbit Hardware

Table 5.2 presents the resource utilization of the Splixbit ICAP controller and the Data Mover. Only 121 slices and 1 BRAM are used. The controller is adapted for the three proposed solutions. For a test bitstream of a CLB-BRAM RM (64 frames + 128 frames + 2 pad frames = 194 frames), a configuration (relocation) latency of 806 µs is obtained. For the same RM with an unencrypted bitstream, the relocation latency would be 197.64 µs (calculated using 194 frames in Table 4.15). As earlier noted (see Section 5.4.1), the ICAP's interface of the 7 series FPGA only accepts encrypted bitstreams at 8 bits per clock cycle as against 32 bits in the unencrypted bitstream. Therefore, the configuration latency has quadrupled compared to that of the CAM's ICAP controller (see Table 4.11).

Table 5.2: Resource usage of the data mover and the ICAP controller

Resource Type	**Component**		
	ICAP Controller	***Data Mover***	***Total***
LUTs	482	974	**1.456**
Flip-Flops	215	1,145	**1,326**
BRAM36s	1	0	**1**
Slices	121	377	**498**

5.5 Evaluation of ATAL's Bitstream Size Overhead

The size of an ATAL-formatted encrypted bitstream increases slightly when compared to the original format from Xilinx. This is because of the additional Use-Encryption words but this increase in bitstream size is negligible when the average size of a PB is considered. Indeed, not more than 108 bytes are incurred for any design bitstream considering that only a maximum of two body parts are expected. Referring to Table

2.11, the minimum reconfiguration frame is 64 for a CLB-DSP pair (plus one pad frame when the MFW is not used) and this is equivalent to 26,260 bytes, not considering user-determined setup commands, which are common to both Xilinx- and ATAL-formatted PBs. The overhead for this CLB-DSP pair is 20 bytes and is only 0.08%. As the design size grows, this overhead is expected to go much lower.

5.5.1 Uncompressed Bitstreams

Table 5.3 shows the maximum ATAL overhead for the various reconfiguration frames of the 7 series FPGA for uncompressed bitstreams. The reconfiguration frame is the minimum selectable resources for PR (see Section 2.2.2). The bitstream sizes (frame data sizes) for the resource block pairs are estimated from the number of configuration frames only without considering the usual preambles, postambles, and the FAR and WCFG command packets, which are always in the bitstream whether or not ATAL is being used for formatting.

Table 5.3: Bitstream size overhead of ATAL for an uncompressed encrypted bitstream

Evaluation Parameters	**Reconfiguration Frames**		
	CLB-CLB	CLB-DSP	CLB-BRAM
Number of Configuration Frames	72 + 1 pad	64 + 1 pad	192 + 1 pad
Number of ATAL Body Parts	1	1	2
Frame Data Size (Bytes)	29,492	26,260	77,972
$ATAL_{ovh}$ (Bytes)	20	20	88
Max Percentage $ATAL_{ovh}$ (%)	0.07	0.08	0.11

The bitstream size overhead of ATAL is determined as the bytes loaded for the Use-Encryption commands (see the local preamble of Section 5.2.2) plus the one *length* word in the global preamble (see Section 5.2.1) and two words each for the IDCODE and RCRC packets (see Section 5.2.3). This implies a total of 18 words (72 bytes) for the first frame and 17 words (68 bytes) for the subsequent frames since the global preamble is included once. However, in the Xilinx-formatted encrypted PB, the Use-Encryption words are also used. That means in an ATAL-formatted bitstream the first

frame actually incurs 5 words (20 bytes). The overhead in bytes, of ATAL ($ATAL_{ovh}$) can thus be described by Equation (5.1):

$$ATAL_{ovh} = 20 + 68 \times (Number\ of\ Body\ Parts - 1) \quad (5.1)$$

5.5.2 Compressed Bitstreams

Table 5.4 shows the ATAL overhead for the various reconfiguration frames of the 7 series FPGA for compressed encrypted bitstreams. When ATAL is used to format a compressed bitstream, the bitstream size overhead is quite different from that of the uncompressed bitstream. Many body parts (at least the number of frame addresses in the bitstream) are expected. At the same time, the local preambles in these body parts are sure to increase the size of the bitstream slightly. This increase is estimated at 68 bytes per frame, which is the same as the overhead incurred per body part, implying a 16.83% increase in bitstream size per frame.

Table 5.4: Bitstream size overhead of ATAL for a compressed encrypted bitstream

Evaluation Parameters	**Reconfiguration Frames**		
	CLB-CLB	CLB-DSP	CLB-BRAM
Number of Configuration Frames	72	64	192
Best-Case Evaluation			
Number of ATAL Body Parts	1	1	2
Frame Data Size (Bytes)	404	404	808
$ATAL_{ovh}$ (Bytes)	20	20	88
Percentage $ATAL_{ovh}$ (%)	4.95	4.95	10.89
Average Percentage $ATAL_{ovh}$ (%)	7.92		
Worst-Case Evaluation			
Number of ATAL Body Parts	72	64	192
Frame Data Size (Bytes)	29,088	25,856	77,568
$ATAL_{ovh}$ (Bytes)	4,848	4,304	13,008
Percentage $ATAL_{ovh}$ (%)	16.67	16.65	16.77
Average Percentage $ATAL_{ovh}$ (%)	16.72		

Regardless of the combination of resource blocks, the maximum $ATAL_{ovh}$ cannot practically exceed 16.77%, which is the overhead for the largest reconfiguration frame. However, in theory, it cannot exceed the per-frame overhead of 16.83%.

5.6 Configuration Strategies for Secure Task Relocation

Having identified earlier, the conflict between keeping bitstreams in cypher format and PBR requiring plain bitstreams, in the following subsections, three techniques that can be used to relocate encrypted PBs are investigated and evaluated. The first involves using a dedicated on-chip custom-built decryptor engine, the second approach involves an initial configuration followed by intermediate readback before relocation, while the third is based on the ATAL approach.

5.6.1 Intermediate Dedicated On-Chip Decryption (IDOD)

In this method, a dedicated on-chip AES decryption circuit is used to first decrypt the bitstream, then PBR is carried out by modifying the plain address in the decrypted bitstream. Bitstream security is not compromised since the decryption is done inside the FPGA. However, a security attack like side-channel analysis [215] can potentially be used to extract the AES key and eventually steal the IP. This is not peculiar to user-implemented decryption circuits, even the Xilinx's internal decryptor is susceptible to this attack as successful attacks on it have shown [216]. Several implementations of AES decryptor have been reported in the literatures. One fitting for this use-case should ideally be resistant to side-channel attack.

The main disadvantage of the method presented here is that the decryptor consumes additional FPGA resources apart from the relocation engine itself. Figure 5.8 shows the configuration and timing model for this method. Apart from the time durations for moving data from the external storage (t_{mov}) and for unencrypted configuration (t_{u_cfg}), time is also incurred for the decryption (t_{dec}). The value of t_{dec} can be reduced at the expense of increased area by pipelining the decryption. The total time required for relocating a PB (t_{pbr}) with this method can be described by Equation (5.2).

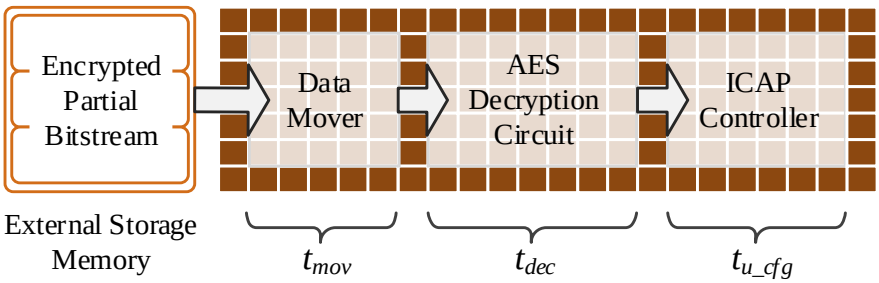


Figure 5.8: Model for relocating encrypted partial bitstreams by using a dedicated decryption circuit for intermediate decryption

$$t_{pbr} = t_{mov} + t_{dec} + t_{u\_cfg} \tag{5.2}$$

5.6.2 Initial Configuration and Intermediate Readback (ICIR)

This involves the initial configuration of an encrypted PB followed by readback and then reconfiguration. Though an encrypted bitstream is used for the initial configuration, the FPGA allows the ICAP to read back unencrypted data. The configuration data can then be written to a different frame address generated in runtime. To ensure that bitstream security is intact, the ICAP signals should never be routed out of the FPGA. In this method, additional time overheads are incurred for the initial configuration and the intermediate readback. Figure 5.9 shows the model for this method while its latency equation can be seen in Equation (5.3). The main contributors to PBR latency here are the times used for the initial encrypted configuration (t_{e_cfg}) and intermediate readback (t_{rbk}).

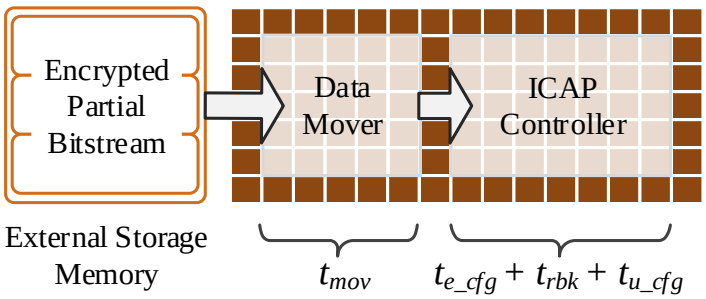


Figure 5.9: Model for relocating encrypted partial bitstreams by initial configuration followed by intermediate readback and final reconfiguration

$$t_{pbr} = t_{mov} + t_{e\_cfg} + t_{rbk} + t_{u\_cfg} \tag{5.3}$$

It should be noted that the idea of relocating an already configured bitstream has already been reported in [147]. However, it was not applied as a means of relocating encrypted partial bitstreams.

5.6.3 Advance Task Address Loading (ATAL)

The configuration model for ATAL is the same as that in Figure 5.9 but for the timing, there are no initial encrypted configuration and intermediate readback times as shown in Equation (5.4). There is only an encrypted configuration after the data movement. One limitation with ATAL has to do with the fact that the ICAP in the 7 series FPGAs only accepts encrypted data on bits 7 to 0 of its 32-bit input port [47], making t_{e_cfg} more than t_{u_cfg}. However, this also affects method 2 since the initial configuration is that of an encrypted bitstream.

$$t_{pbr} = t_{mov} + t_{e\_cfg} \tag{5.4}$$

5.7 Evaluation of the Configuration Strategies

In order to quantitatively compare the three methods for relocating encrypted PBs, uncompressed and compressed bitstreams are generated for a 2.4-GHz wireless communication application and used as the basis for determining the relocation latencies. Both plain and encrypted PBs are generated using Vivado. The application's RM/RP amounts to a CLB-BRAM pair which is equivalent to 194 configuration frames (64 CLB/INT frames, 128 BRAM frames, and 2 pad frames). With 101 32-bit words in each frame of the 7 series FPGA, the unencrypted configuration data for the application is a total of 19,594 32-bit words, not considering the configuration commands.

In the analyses that follow, it should be noted that the data movement time (t_{mov}) has not been considered since it is the same (3.993 ms, measured using the Zynq processor's internal timer) for all the three methods. There are slight differences in the lengths of the final bitstreams. However, with respect to the transfer time by the Data Mover, all the bitstreams can be moved into the FPGA in the same amount of time. This is because of the contiguous nature of the data movement.

5.7.1 Evaluation of IDOD

In terms of area utilization, it is certain that this method uses the most resource because of the need for a decryption circuit. The other two methods simply require the Data Mover and the ICAP Controller. From the work in [217], we estimate a resource usage of 5081 slices for an AES-256 decryptor targeted at the 7 series FPGA. When this usage is combined with the usage for the Data Mover and the ICAP Controller, the total usage comes up to 5579 slices.

In order to estimate the decryption time, t_{dec}, the decryption latency of 41 clock cycles (197.5 ns) per block reported in [217] is used. The encrypted portion of the encrypted bitstream of the test application amounts to 19,784 32-bit words, which is equivalent to 4,946 AES blocks (with a block being 128 bits). This is different from the unencrypted word count because padding words are used before authentication and encryption to ensure that data lengths are a multiple of 128 bits for encryption and 512 bits for authentication, required for simplifying the MAC computation [43]. Assuming no pipelining is used, we estimate a total decryption latency of 4,946 multiplied by 197.5 ns, that is, 976.84 µs. This would reduce with pipelining, but the reduction would depend on the pipeline depth.

For an unencrypted frame data, the configuration time of the Splixbit ICAP controller is the same as that of the one in Chapter 4. Therefore, from Table 4.15, the configuration time is computed as 197.64 µs and from Equation (5.1), t_{pbr} can be estimated as 1,174.48 µs. The comparison of this with those of the other methods is presented in Table 5.5.

5.7.2 Evaluation of ICIR

The resource overhead incurred with this method is that of the Data Mover and the ICAP Controller, which is a total of 498 slices, and is much lower compared with that for IDOD (see Table 5.5). For the 194 frames required for the application, from Table 4.15 and real-life measurement, the estimates for t_{rbk}, t_{u_cfg}, and t_{e_cfg} are 195.4 µs, 197.64 µs, and 806 µs (measured) respectively. This totals 1,199.04 µs for t_{pbr} according to Equation (5.3). This is the worst relocation latency of the three methods.

5.7.3 Evaluation of ATAL

Like ICIR, the resource usage here is only 498 slices, which represents a resource saving of more than 90% compared to IDOD. It should be noted, however, that the ICAP controller includes a readback functionality which is not needed in ATAL. Removing this functionality would reduce the usage by a few slices. For the relocation latency estimation, a value of 806 µs for the application's RM was measured. This value is the lowest of all the three methods, offering an average time saving of 32% over the IDOD and ICIR methods. This is because only a single configuration is done, with no intermediate decryption or configuration and readback, making ATAL the most resource and time-efficient of all the three.

The t_{e_cfg} for ICIR and ATAL are as high as they are because encrypted bitstreams can only be configured 8 bits per clock cycle. This is a limitation in the 7 series FPGA and has been addressed in the UltraScale, where the entire 32-bit interface can be used [218]. Applied to the UltraScale then, all the other values in Table 5.5 would be the same apart from t_{pbr} for ICIR and ATAL methods, which would reduce to estimated values of 594.54 µs and 201.5 µs respectively, but resulting in time savings for ATAL of 83% and 66% over the IDOD and ICIR methods respectively.

Table 5.5: Comparison of the three methods for resource usage and relocation latency

Parameter	**IDOD**	**ICIR**	**ATAL**
Utilization	5579 slices	498 slices	498 slices
Relocation Latency, t_{pbr}	1,174.48 µs	1,199.04 µs	806 µs

5.8 The Security Implications of ATAL

It is important to consider the security implication of using ATAL secure PB format. It used to be that only the FDRI data of the bitstream was encrypted in the Virtex-5 secure bitstream format [43]. On the contrary, in the 7 series FPGA, only the Use-Encryption commands are in plain format. In the ATAL format, the only plaintext content that reveals design-related information is the number of body parts, which reveals the number of resource types used and nothing more. That is all that the attacker can glean

about the design and the author reckons that until proven otherwise, not so much can be done with this information in terms of stealing the design or attacking the device.

To understand what the attacker would know from the number of body parts, refer to Table 2.2, and note that the CLB, I/O blocks and clock routing resources share the same block type of '000', while BRAM content uses the block type of '001'. The Xilinx documentation does not indicate the block type for DSPs, but an inspection of bitstreams generated in Vivado reveals that the DSP shares the same block as CLBs. When the bitstream compression feature is not used, the only frame addresses in the bitstream are those for the combination of resource blocks used by the design. In such bitstreams, a maximum of two frame addresses (for BRAM blocks and CLB/DSP/IO/CLK blocks) and thus, a maximum of two body parts, can be expected.

Moreover, while packing the bitstream parts into a single bitstream file, plain FAR command and frame addresses can be included right in the local preambles to simplify the design of the Splixbit hardware and bitstream loading. However, this would give far too much information away about the design. Once a specific FAR value is known, an attacker with a fair understanding of the FPGA architecture would know exactly what resource is used and where it is on the chip. The device is then more prone to sophisticated attacks like probing the chip with a scanning electron microscope [43]. As such, it is far more secure to specify the FAR value in runtime.

5.9 Chapter Summary

PBR is a technique that can be used in FPGAs to reduce external bitstream storage requirement and facilitate adaptability. However, because of the recent increase in the uptake of FPGAs, bitstream authentication and encryption have been introduced to secure FPGA-based IPs. PBR meanwhile, can only be carried out on plain bitstreams. While method fragments already exist that can be used to effect encrypted partial bitstream relocation, these are far less time- and resource-efficient. On the one hand, an on-chip decryption engine can be used to decrypt the bitstream before passing it on to the configuration interface, in which case, the difference in resource utilization brought about by ATAL is the same as the utilization of the decryption engine which can almost

occupy an entire chip (for smaller devices). On the other end of the spectrum, an initial configuration followed by an intermediate readback and final configuration can be used, with the consequence that a very large system time overhead can be incurred in a bid to save resources. ATAL provides an efficient solution and it is not a middle-ground approach that incurs moderate resource and time overheads; it offers the best of both spectra, incurring a very negligible time and resource overhead. ATAL has been compared with these other methods and it happened to provide resource and time savings in the order of 90% and 80% respectively for the test case.

Chapter 6

6

FPGA Clock Infrastructures for Dynamic On-Chip Inter-Task Communication

The importance of on-chip communication in today's System-on-Chip (SoC) platforms cannot be overemphasized. With the continued increase in the density of chips, the number of circuits that can coexist on a single chip has increased by astronomical proportions, sometimes numbering in the thousands [219]. Some of the processing elements or circuits that can coexist on a chip include CPUs, GPUs, DSPs, memory elements, and other IP modules. Establishing inter-circuit data transfer among these circuits in a scalable and resource-efficient manner is essential. In an FPGA, the area utilization of a circuit is influenced by the *routability* of the resources in the area occupied by the circuit [220]. Underutilization can occur if the resources in a region of the FPGA cannot be used by a circuit because there are no interconnect resources or possible routes to connect them, leading to an enlargement of the area occupied by the circuit in order to encompass more routing resources. The result is that the circuit occupies a larger area than necessary, with several unused, or rather, unusable resources.

The NoC has come to be regarded as the future of on-chip communication, owing to advantages such as modularity and concurrency [173] (refer to Section 3.4). When implemented on an FPGA, in order to provide access to the communication network, the NoC typically uses the general routing resources as network links, thus increasing the demand on the already strained routing resources in the area occupied by the circuit. To alleviate this problem, a network link that does not use the general interconnect resources should be used where possible. Incidentally, it turns out that most FPGA-based designs do not use the on-chip global and horizontal clock buffers [44] and invariably, a large part of the clock network. Repurposing these buffers and networks for use as network links would lessen the demand on general routing resources for inter-

circuit communication. Moreover, these otherwise redundant resources, which are not used for their intended clocking-related purposes have already been paid for in silicon. As such, using them for communication-related purposes represents an added value.

Another advantage of using clock buffers and nets for communication is that they facilitate runtime PBR. One important challenge with PBR, which has limited its applicability, is the provision of dynamic communication for relocatable circuits in runtime as previously discussed in Section 3.2.5. This is because inter-circuit links are statically determined at compile time. Runtime routing is a possible solution to dynamic communication but it is both complicated and computationally expensive, often requiring several thousands of clock cycles [151]. In traditional NoCs, the general routing resources are used as network links and thus, constitute static routes and hamper circuit relocation. The use of clock buffers as network links avoids the restriction of static routes and allows the arbitrary relocation of circuits. Since the clock buffers, trees, and nets do not use the general logic routing resources [44] but have their dedicated routing in the clock layer as explained in Section 2.1.1, the path from a transmitting circuit to a receiving circuit is free of general routings.

In addition, routing congestions are often the reason that static routes cross into RPs and requiring that these routes be preserved in all RMs using the RP, which is a further requirement of PBR (see Section 3.2.5). A way of reducing routing congestion, especially at the interface of circuits, and thus, reducing the number of static routes crossing RPs is to use bit-serial interconnections between circuits, as this has been shown to have reduced footprint and congestion factors [180]. Incidentally, our use of clock buffers for communication calls for the adoption of bit-serial connection at the clock buffer level. However, multiple bit-serial connections can be used depending on the availability of clock buffers in the target FPGA. In addition, a bit-serial communication implementation nevertheless is beneficial because it helps in easily meeting the requirement for the preservation of existing static routes, while at the same time garnering the other benefits of bit-serial over bit-parallel interconnects, which include high speed and power savings as demonstrated in [179] and [180].

Because this technique incurs a low overhead of resources and involves the unique use of clock buffers for serial network interconnection, we have termed it *Clock-Enabled Low-Overhead Communication* (CELOC).

The techniques and implementations reported in this chapter are covered in publications [221], [222], [223], and [224]:

- **A. Adetomi**, G. Enemali, and T. Arslan, 'Clock Buffers, Nets, and Trees for On-Chip Communication: A Novel Network Access Technique in FPGAs', in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 219–222.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Relocation-Aware Communication Network for Circuits on Xilinx FPGAs', in *2017 International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.
- **A. Adetomi**, G. Enemali, and T. Arslan, 'Characterization of Clock Buffers for On-Chip Inter-Circuit Communication in Xilinx FPGAs', in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.
- **A. Adetomi**, G. Enemali, G. Seetharaman, and T. Arslan, 'Fault-Tolerant Mechanisms for Relocation-Aware Dynamic On-Chip Communication on FPGAs', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 214–217.

6.1 Clocking Resources in the Xilinx 7 Series FPGA

There are several clock buffers in the Xilinx FPGAs for driving clock networks that span diverse clusters of regions. There are also *Phase-Locked Loop* (PLL) frequency synthesizers for on-chip jitter-free clock generation from an external clock source.

6.1.1 Clock Buffers and Network Distribution

The clock network of all modern FPGAs is based on the *Spine-and-Ribs* topology [225], where vertical spines drive clock signals into horizontal ribs. Eventually, local ribs in the clock regions directly clock logic resources. This provides support for multiple local

and global clock domains. Figure 6.1 shows the clock network distribution and interaction in a single clock region of the 7 series FPGA. There is a horizontal clock row (HROW) that spans the entire length of the clock region. It is in the middle of each region. Clock signals switch vertically upward and downward from the HROW to reach logic resources.

A regional network serves synchronous resources contained in a single clock region and is driven by regional buffers (BUFRs) while a multi-region network distributes clock signals to multiple clock regions, vertically through multi-region buffers (BUFMRs), or horizontally through horizontal clock buffers (BUFHs). A global clock network spans the entire device and can drive multi-region and regional networks. This network is driven from the centre of the device by 32 BUFGs, with 16 in each top/bottom half. The most important thing to note, and that which is being exploited in our adaptation of clock buffers and networks for inter-task communication is that the clock networks use independent physical wires different from the general logic routing. Inside a clock region, switch matrices route clock signals to the logic resources with appropriate PIPs activated as required.

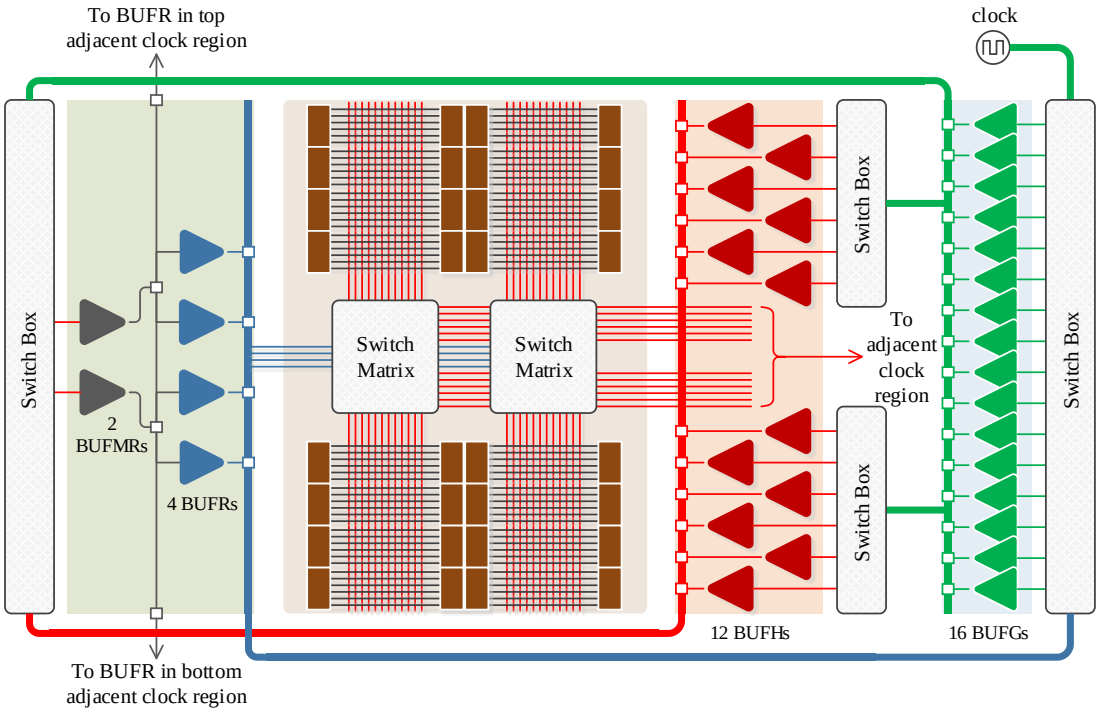


Figure 6.1: Clock network distribution in the clock region of an FPGA

6.1.2 Clock Buffers and the Features Exploited by CELOC

In order to realize CELOC, two factors are important for a clock buffer and its associated net – the span or reach of the net, and the availability of a *Clock Enable* (CE) pin on the buffer for logic functions. The reach or range of a buffer's net determines how far on the chip the clock-layer-based communication signal can travel, and the number of buffers with switchable CEs affects the number of transmitting nodes a CELOC-based network can support. In addition, the clock buffer must be user-accessible in the design tool, that is, it must be possible to instantiate and place it in order to control connections to and from it. We now consider the case take a look at these features in the clock buffers of the 7 series FPGAs [44]:

A. Global Clock Buffers/Multiplexers - BUFGCTRL

The global clock buffers drive the global vertical clocking backbone in the 7 series device and there are 32 of them per device, with 16 in each of the top and bottom halves. Their reach spans the entire FPGA and as such, they can feed any clocking point in the device. As such, they can be used for device-wide communication. These buffers do not reside in clock regions but up to 12 unique lines out of the 32 global clock lines can be driven into a clock region to feed clocking points inside the region. There are six possible configurations of the global buffer when placed in a site on the chip. However, the two of interest are BUFG and BUFGCE (see Figure 6.2)

The global clock nets have the capacity to drive not only the clock inputs of logic resources, but also the Set/Reset (SR) and CE inputs of registers. This feature is particularly important in achieving CELOC-based dynamic communication, as it allows a communication clock signal to be received via the SR input of a register, and ensuring that no local (static) routing crosses the task boundary.

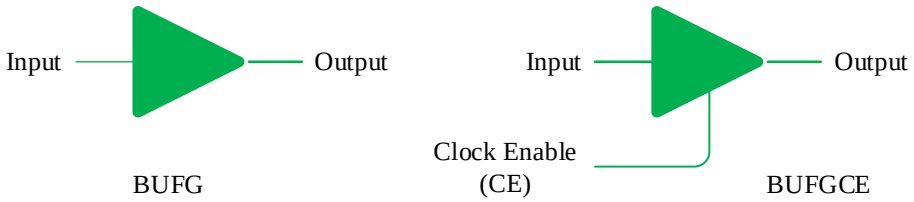


Figure 6.2: Xilinx FPGA's global clock buffers/multiplexer

B. Horizontal Clock Buffers – BUFH and BUFHCE

They are very similar to the global buffers as they drive the horizontal global clock tree spines. However, they span only two horizontally adjacent clock regions. Unlike global clock buffers that do not reside in clock regions, there are 12 BUFH(CE)s contained in the HROW at the edge of each clock region. However, both share the same 12 routing tracks in the HROW. The horizontal buffers can be used as BUFHs, with simple input and output, or as BUFHCEs, with a CE pin (like the BUFG(CE)s of Figure 6.2). Unlike the BUFGCTRL, the CE of the BUFHCE can be used to gate power consumption and as well achieve a true CE logic function on a clock cycle-to-cycle basis, allowing both synchronous and asynchronous transfer of the clock input to the output of the buffer. Like the global clocks, the BUFH(CE)s can drive the enable and reset inputs of logic resources.

C. Multi-Region Clocks – BUFMR and BUFMRCE

The multi-region clock buffers are used to enable multi-regional clocking by directly driving regional clock buffers (BUFRs) in the same clock region and the ones above and below it. Like the BUFHCE, the CE in the BUFMRCE can be synchronous or asynchronous to the input/output transfer. The BUFMRCE can be used to achieve a CELOC-based network that is local to three vertical clock regions. There are two BUFMR(CE)s in each clock region.

D. Regional Clock Buffers - BUFR

The regional clock buffers can drive any clocking point drivable by a global clock in a single clock region. In each region there are four dedicated clock nets driven by four BURs. These nets are independent of each other and the global nets in the clock region. Since each of these regional clock nets is distinct, this allows multiple unique clocks to feed a single design or provide communication. The regional clock buffers can be used in both BUFR and BUFRCE configuration. They have two control lines, the CE and the clear (CLR). These are associated with the frequency division mode of the buffer and can only be used in that mode. That is, the CE can only be toggled if the BUFR_DIVIDE option is set to any number other than “BYPASS” when the buffer is instantiated in RTL. With respect to CELOC, the BUFRs can be used for intra-region

communication and are also essential for transferring data out of a clock region for global inter-task communication as they are able to connect directly to BUFGs.

6.2 Adaptation of Clock Buffers for Communication

The availability of a diverse range of clock buffers with global and local spans in the Xilinx FPGAs offers a unique possibility that can be utilized to achieve on-chip communication functionality. Our communication solution involves a special adaptation of these clock buffers to serve as binary (‘0’ or ‘1’) signal transmitters and receivers on the FPGA.

Would repurposing clock buffers for communication not be detrimental to their intended functionality? It is true that the clock buffers and nets are precious and are available in the chip predominantly for clocking-related functionalities. For instance, according to the 7 series FPGA clocking resources user guide [44], the functions of the buffers include glitchless multiplexing between clock sources, clock gating to reduce dynamic power consumption, elimination of clock distribution delays, clocking support for circuits spanning multiple clock regions on the FPGA, and clock frequency division. However, the same user guide reports that most FPGA designs contain several unused global and horizontal clock buffers. These buffers are essential to the working of CELOC. One of the aims of this research is to repurpose these redundant resources for dynamic on-chip communication support and thus save on valuable FPGA resources that would otherwise have been used for on-chip networking, while at the same time providing a static-route free inter-communication for relocatable circuits.

Figure 6.3 presents the CELOC concept in a diagrammatic form. By gating a free-running communication clock using a clock buffer, it is possible to send data from a transmitting (TX) task to a receiving (RX) task from any location on the device to another reachable by the buffer. At the TX end, a *Serializer* works in a *Parallel-In Serial-Out* (PISO) version to send data while a *Deserializer* at the RX reverses the operation in a *Serial-In Parallel-Out* (SIPO) version. The CELOC technique requires an RX task to be fed with three clocks: *task_clock*, *com_clock*, and *data_clock*. The

task_clock signal is used to clock the tasks while *com_clock* is used to generate data_clock, which carries a serialized data from the source to the destination.

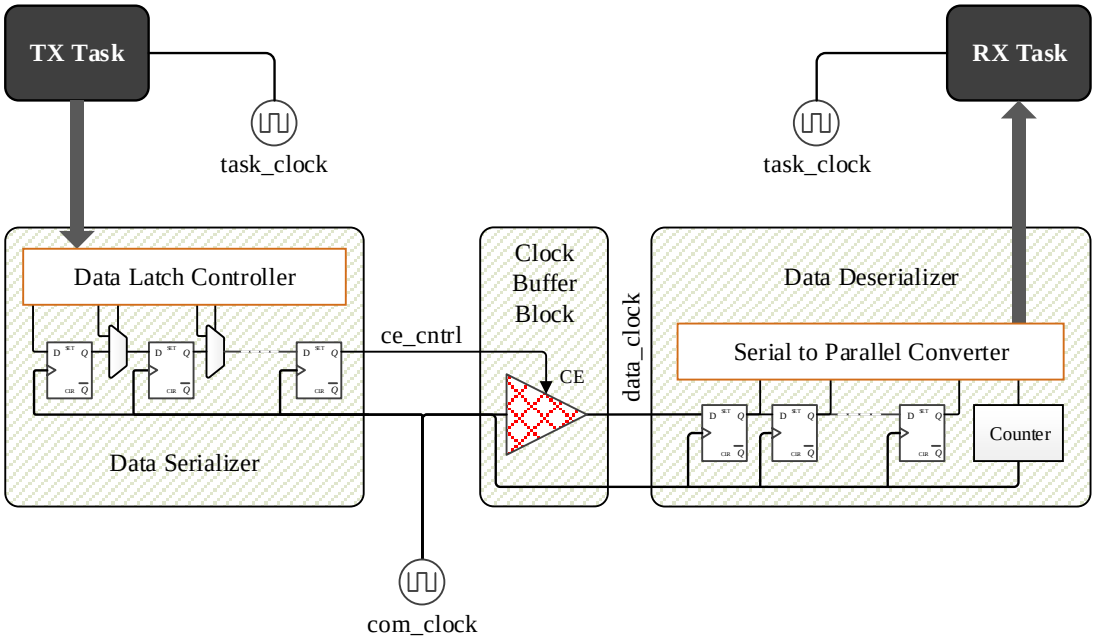


Figure 6.3: Transmitting serialized data with a clock buffer

6.2.1 Data Transfer Mechanism

To transfer data, the parallel data from a TX task is serialized and shifted out bit-by-bit to an RX task through the clock buffers. A register is used to latch the parallel data for onward shifting to the clock enable (CE) of the buffer on the ce_cntrl signal line. This latching is done by the *Data Latch Controller*. Since the same register block is used for shifting out the serial bits, multiplexers are used to select between updating the registers with new data and shifting already latched data.

The *ce_cntrl* signal, which carries the serial data to be transmitted controls the output of the buffer by toggling its CE. A '1' allows the input of the buffer to pass through to the output, while a '0' ties the output to zero. Since the communication clock (which can be the same as the task clock) and the task clock are synchronous, a '1' on ce_cntrl essentially allows a full clock cycle to pass through while a '0' blocks it. As an example, Figure 6.4 shows the theoretical expected signal transitions for transmitting 10011010 (binary) (see Table 6.1 for the corresponding truth table). The RX task's

SIPO circuit can detect a falling edge on com_clock as a '1'. With respect to the distance between the TX and RX tasks, the clock buffers in the Xilinx FPGAs are designed for short propagation delays and very low skew [44]. This helps prevent the kind of long propagation delays associated with shared-bus interconnects. As a result, the two clock signals (com_clock and data_clock) can travel far with minimal loss of phase alignment, and thus ensure timing closure.

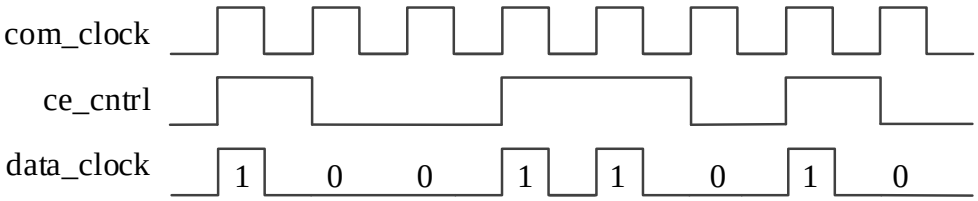


Figure 6.4: Transmission of an 8-bit binary data 10011010

Table 6.1: Truth table for clock-enabled data transmission

Inputs		**Outputs**
tx_serial_data (CE)	***com_clock***	***data_clock***
1	X	com_clock
0	X	0

6.2.2 Communication Clock and Task Clock Generation

In order to achieve the maximum possible throughput for data transfer, it is important to drive com_clock as high as possible. An advantage of using a separate clock as the communication clock is that we are not limited to the frequency of the task clock; the communication engine can run at a much higher frequency. The FPGA provides a hard PLL clock generator, which can be used to generate a clock signal at a frequency much higher than that of the clock fed into the FPGA.

In the demonstration of CELOC in this work, the PLLE2_BASE primitive in the 7 series FPGA is used to generate the two clocks (com_clock and task_clock). Two global clock buffers are then used to distribute them throughout the chip when necessary. Figure 6.20 shows the schematic of the PLL-based clock generation and

distribution for CELOC. The core of the clock generator is the PLLE2_BASE primitive, which can be used as a frequency synthesizer, jitter filter, or to deskew clocks. As a clock generator, the PLL requires an internal feedback as shown in Figure 6.20.

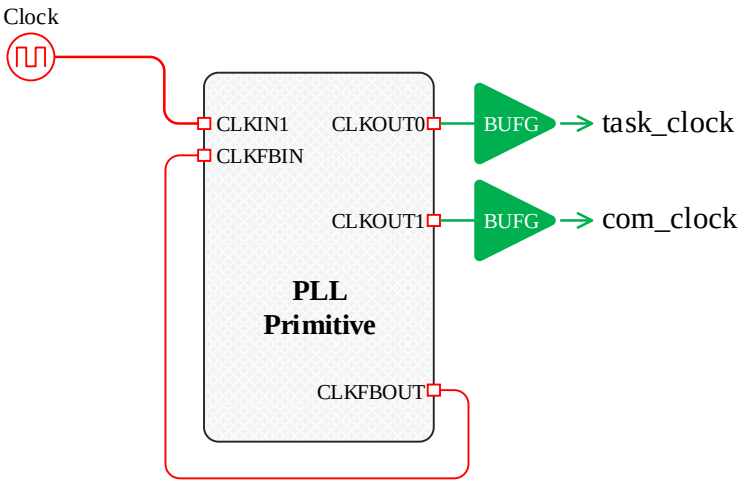


Figure 6.5: Schematic of the PLL-based clock generator for CELOC

6.2.3 Clock Domain Crossing

Because com_clock and task_clock are functionally in different clock domains – one used to clock the registers that push out the serial bits, and the other to retrieve the serialized data through another set of registers, it is important to investigate the impact of *Clock Domain Crossing* (CDC).

To avoid complications from CDC, the two clocks are sourced from a single PLL clock generator with the communication clock made as high as possible. Using this structure helps to prevent setup and hold timing violations by keeping both the transmission and the reception synchronous and in the same clock domain – no asynchronous clocks and no variable phase alignment. Therefore, no clock domain crossing issues are expected since the same clock (com_clock) is used to transmit and receive data [226]. Nevertheless, every implementation of CELOC is checked for CDC violations using the Vivado timing report.

6.2.4 Data Recovery Mechanism

Since we are interested in avoiding the general routing resources, it is important that the recovery of the serial bits in the RX circuit should employ a mechanism that is independent of general interconnects. Hence, an ideal interface to data_clock should be a clocking point in a logic element. Two candidates for this are registers and latches with non-clocking inputs that can be fed by clock signals. The FDPE register and the LDPE latch [227] in the 7 series FPGA fall into this category and can thus be connected as shown in Figure 6.6 to receive data_clock into an RX circuit without using the general interconnect. This is because their Set/Reset (SR) and Preset inputs can be driven by global and horizontal clock buffers. Their Q outputs produce the same waveform as the original ce_cntrl signal used to toggle the clock buffers in Figure 6.3. The choice of either the FDPE or the LDPE influences the maximum bandwidth of communication as will be seen in Section 6.5.

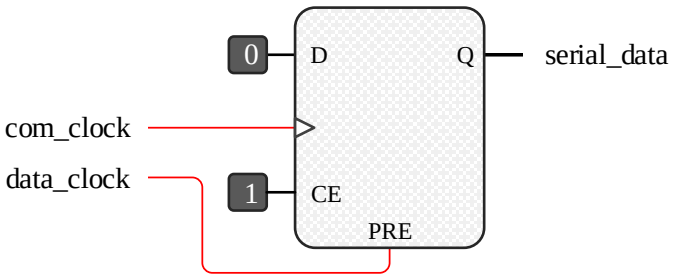


Figure 6.6: Setup of the FDPE (or LDPE latch) register to interface with com_clock and data_clock for serial data recovery

Table 6.2: Truth table of the FDPE register

Inputs				**Outputs**
PRE	***CE***	***D***	***C***	***Q***
1	X	X	X	1
0	0	X	X	No Change
0	1	D	1	D

The FDPE is a D flip-flop with clock enable (CE) and asynchronous preset [227]. By connecting CE to a ‘1’ and D to a ‘0’, with the clock input fed by the same clock

(com_clock) used to create the data clock at the transmitter, data_clock connected to the PRE input produces on Q, signal level transitions corresponding to the rising edges of data_clock as shown in Figure 6.7 for the same 8-bit data 10011010 (binary) transmitted in Figure 6.3. To understand how this works, we consider the truth table of the FPDE (see Table 6.2). We observe that by setting CE to '1' and D to '0', Q follows PRE (data_clock) instead of D at every rising edge of C. The LDPE data latch with asynchronous preset and gate enable [227]. A similar explanation applies to the LDPE with regards to the signal transmissions that allow the recovery of the transmitted data, except that for the LDPE, the gate (connected to data_clock) input's signal transitions are reversed.

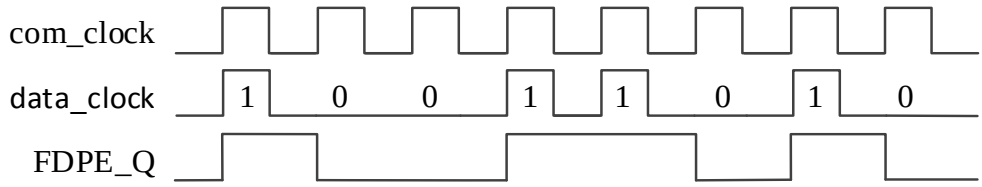


Figure 6.7: Waveform showing the signal transitions at the output of an FDPE register with com_clock as the clock input and data_clock as the PRE input

6.3 Packet Synchronization and Encoding

Since CELOC in general, serializes the data being sent before transmission, an idle line will be either a '1' or a '0'. It then becomes important for the transceiver task to determine when to start or stop reception? It is also possible for a task node to join a CELOC-based network in the middle of an ongoing transmission. The new node has to correctly latch on to the beginning and end of packets. In general, a CELOC-based data transfer between any two circuits does not require any special handshaking or encoding technique, as a data bit that leaves the source circuit would arrive at the destination circuit without any ambiguity if the two circuits are directly interconnected. However, in CELOC-based NoCs, the source and destination nodes may not be directly physically connected and communication packets may be routed through intermediate nodes until they reach their destinations.

As such, data packet synchronization may be required to coordinate data transfer. It may be worth nothing however, that packet synchronization is not always required in bit-serial networks [180]. Depending on the adopted topology, a NoC design might be able to do away with packet synchronization and as such save on encoding resources and latency. This approach is favoured when applicable. However, in other applications, encoding is necessary if ambiguity in data transfer is to be avoided

To uniquely mark off the boundaries of transmission packets in serial networks, frame synchronization mechanisms are used. One such mechanism is byte stuffing, where a special code byte is used to delimit packet boundaries. In order to prevent incorrect synchronization, as the code byte may be present in the data packet, special 'escape' codes are often used [228], but the length of the packet ends up being inconsistent [229]. This is not desirable in real-time applications, where timeliness and predictability are important. To achieve consistency in packet size, we propose an adapted form of the *Consistent Overhead Byte Stuffing* (COBS) [229].

The COBS maps numbers in the range [0, 255] to numbers in the range [1, 255], thereby reserving one number which can be used as the frame synchronizer (delimiter). The details on how this is achieved can be found in [229]. Adopting a similar technique to map the hex number set [0, F] to [1, F], we reserve the number zero to be used as the delimiter. We call this *Consistent Overhead Nibble Stuffing* (CONS). Starting at the zeroth (most-significant) nibble (4 bits of 0's and 1's), the occurrence of a zero is replaced by the number of nibbles examined (including the zero) followed by the non-zero nibbles before the zero. For example, an arbitrary 32-bit packet of hex numbers (400AD013) passed through the CONS encoder would produce 2413AD313 (hex) as shown in Table 6.3.

A simple way to carry out the encoding is to logically pad the packet with zero nibbles at the beginning and end as shown in the second row of Table 6.3, with the first serving as a placeholder for the overhead and the other as a phantom helper to complete the encoding process. This phantom does not actually count as part of the data. Each nibble of the padded packet is then given an index starting at 0 from the most significant nibble (0 to 9 in this example). The encoded nibble of a zero nibble at index i_{zn} is obtained by subtracting i_{zn} from i_{znext}, where i_{znext} is the index of the next zero nibble.

There cannot be another nibble after the appended zero nibble. Therefore, the encoded packet is terminated on the penultimate index (index 8 in this example). Further illustrations in Table 6.3 show that an all-zero packet would be encoded as 111111111 (hex) and a packet without a zero nibble as 9XXXXXXXX (hex), where X is a non-zero nibble.

The advantage of this form of encoding is that every packet is guaranteed to have a fixed overhead of one nibble. On the other hand, the disadvantage, as the examples show is that even when there is no zero nibble in the data, the overhead is still incurred. However, this is the price that is paid for the benefit of determinism in communication latency as far as the data packet is concerned.

Table 6.3: Examples showing the CONS encoding process

Index (i_n)	0	1	2	3	4	5	6	7	8	9
Nibbles (D_i)	0	**4**	**0**	**0**	**A**	**D**	**0**	**1**	**3**	0
CONS Code ($i_{znext} - i_{zn}$)	2		1	3			3			
Encoded Data	**2**	**4**	**1**	**3**	**A**	**D**	**3**	**1**	**3**	
Nibbles (D_i)	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
CONS Code ($i_{znext} - i_{zn}$)	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	
Encoded Data	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	
Nibbles (D_i)	0	**5**	**1**	**D**	**F**	**2**	**C**	**3**	**7**	0
CONS Code ($i_{znext} - i_{zn}$)	9									
Encoded Data	**9**	**5**	**1**	**D**	**F**	**2**	**C**	**3**	**7**	

In order for a new node to synchronize to the communication network, a bit-level framing delimiter is required. Since there is no zero nibble in the CONS encoding, there cannot be more than three consecutive 0's except if a zero delimiter is used. To avoid ambiguity, a sequence of 1 and seven 0's (10000000) will be used as the delimiter taking a cue from [229]. This delimiter or *Frame Synchronization Sequence* (FSS) is added at the beginning of each transmission packet.

The FSS, the CONS overhead, and the data bits can all be concatenated into a single packet as shown in Table 6.4. The data bits can be no more than 7 bytes long for

a fixed-width data packet as this is the maximum number of bytes between any two successive zeros that can be encoded by the CONS scheme. This is because the 16 nibbles in the set [0, F] are mapped to 15 nibbles in the set [1, F] for the purpose of encoding zero nibbles. In other words, only 15 consecutive zero nibbles (60 bits) can be encoded (including the phantom zero appended logically to carry out the encoding). Removing the 4 bits of this phantom zero leaves 56 bits (7 bytes) for the actual data to be transferred.

Another way of looking at this is to note from Table 6.3 that a code nibble is derived from the subtractive operation between the indexes of two nearest zeros, with the code stored in the position of the first (placeholder) zero nibble. Since a nibble can only hold a maximum count of 15 (F in hexadecimal), and the phantom zero appended to the data is part of the count, the actual data (payload), which is the allowed maximum value of the distance between any two zero nibbles, is thus 14 nibbles (index $15+i_n$ minus index i_n minus one phantom overhead nibble). This 14-nibble maximum data bits is only true for packets with infrequent zeros; if zeros are guaranteed to show up at no more than 14 nibbles apart, then a single packet can have data bits in excess of 56 bits. However, where such guarantees are not deterministic or where bounded transmission latencies are desired as is the case of real-time on-chip networking, a fixed data bit of not more than 56 bits has to be enforced.

Nibbles have been used instead of bytes as a compromise between the percentage overhead and the maximum data bits. With a byte word length as in the original COBS, we would incur 8 bits of overhead per packet, though the maximum data bits would then be 254 bytes. A quick comparison shows that the COBS has a lower percentage overhead of 0.39% per 254 bytes compared to 7.14% per 14 bytes in CONS. However, at lower data sizes, COBS incurs more than CONS. For instance, for a data size of 6 bytes, COBS would incur 20% overhead compared to 8.33% in CONS. Moreover, the size of the delimiter also increases with the word size, always two times the word size, and thus influencing the total overhead and latency of packet transactions. Ultimately, the choice of word size will be a compromise between the percentage overhead and the maximum data bits required per packet.

Table 6.4: Packet format for CONS-encoded data bits

Fields	***FSS***	***CONS Overhead***	***Max. Data***
Number of Bits	8 bits	4 bits	56 bits
Comment	Value: 80 hex	CONS-encoded	

6.4 Network Adapter for Communication Access

To exploit the clock network for communication, each of the intercommunicating tasks in an RC system employing CELOC must be wrapped with a *Network Adapter* to arbitrate access to the CE of a clock buffer, When no packet synchronization is used, the CONS encoder and decoder are not needed and adapting to a network simply requires the Serializer/Deserializer (SERDES) of Figure 6.3 introduced in Section 6.2.

The popular serial communication interfaces like the *Serial Peripheral Interface* (SPI) and I^2C are avoided because they require more than one signal. A potential interface protocol for CELOC could be a 1-wire protocol like the one introduced in [230] or the *Universal Asynchronous Receiver-Transmitter* (UART). Essentially, since only the CE pin in a clock buffer is being driven by an RX task in CELOC, a single-wire protocol would be more appropriate to prevent the usage of static routing resources as much as possible. The proposed SERDES provides a raw interface to the clock buffers and a higher-level bit framer can always be used to adapt to different serial protocols. As it is, the SERDES is a serial streaming interface that would bit-stream a packet of data presented at its data input and also recover a parallel data that is serially shifted in.

On the other hand, when packet synchronization is needed, the CONS encoder and decoder are used and the task is wrapped as shown in Figure 6.8. This builds upon the proposed SERDES by implementing five major blocks: a CONS Decoder, a *Task Interface Logic* (TIL), and the CONS Encoder. The next subsections provide more details on these blocks and other components of the network adapter.

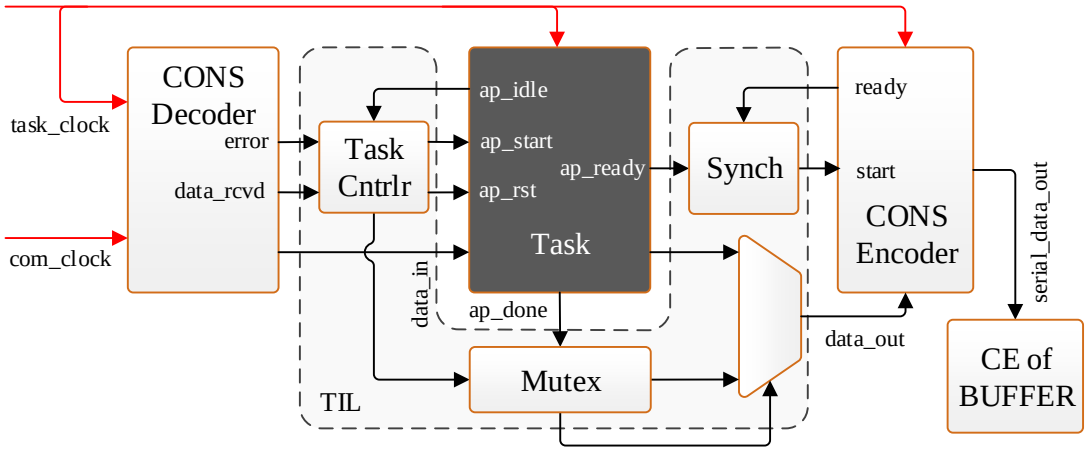


Figure 6.8: Network adapter for packet-synchronized communication access in CELOC

6.4.1 Task Interfacing

This work proposes a task interface model that is based on the Xilinx HLS Block-Level interface protocol [15] for any task that has to communicate using CELOC's packet-synchronized wrapper. This model requires that a minimum of five ports: *ap_idle*, *ap_start*, *ap_rst*, *ap_ready*, *ap_done*, and *ap_return* (indicated by the arrow that feeds the Mux in Figure 6.8) are defined for a task. This ensures uniformity of interfacing between different tasks and the CONS codec (decoder-encoder) and provides a standardized task interface. In addition, this is also in line with the current trend in using HLS-generated HDL modules for rapid system development.

6.4.2 CONS Encoding

In the encoder, which also serves the function of data serializer, the CONS encoding algorithm is implemented with a finite state machine. The flowchart in Figure 6.9 is a representation of the encoding process. The thick junctions depict the concurrency of the implementation. The theoretical encoding process presented in Section 6.3 is modified for hardware implementation. The encoder starts its operation when a START signal is asserted. First, it saves the data to encode in a shift register and then starts the encoding process. The process involves detecting zero nibbles and replacing them with CONS codes. Once the entire packet is encoded, the bits are shifted out serially for routing to the CE of a clock buffer. The encoder asserts a READY signal when all the bits have been shifted out, signalling that it is ready for another encoding operation.

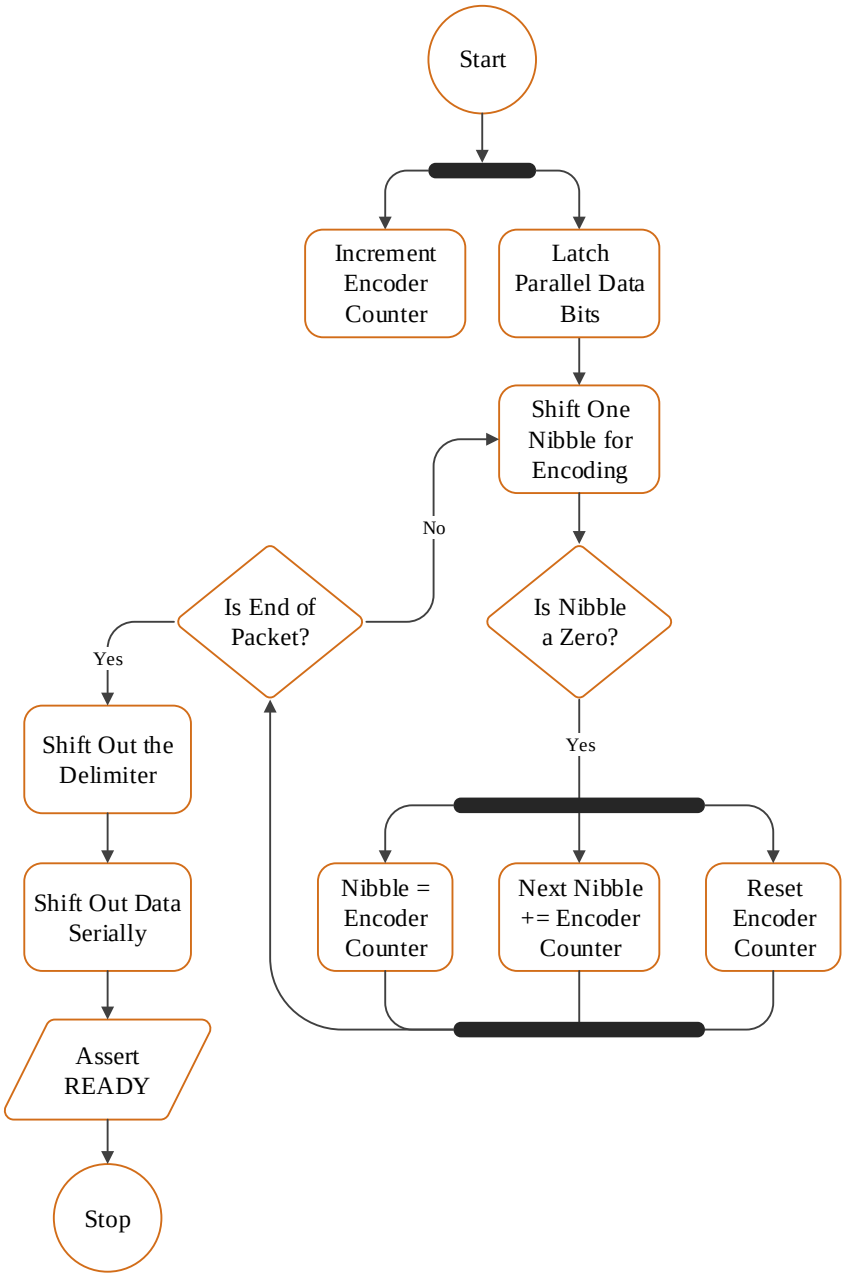


Figure 6.9: Flowchart describing the CONS encoder's implementation

6.4.3 CONS Decoding

The CONS decoder receives, decodes, and de-serializes a CONS-encoded packet. The decoding process can be roughly represented by the flowchart in Figure 6.10, where the thick junctions depict the concurrency of the implementation. In the decoder, the code nibbles in the received packet are replaced with zeros. The decoding is simplified by a careful implementation of the decoding algorithm. A close look at the encoded data in

Table 6.3 in reveals that every code nibble points to the relative location of the next code nibble. This is as expected since the CONS codes are formed by counting the number of non-zero nibbles preceding a zero nibble as explained in Section 6.3. Also, the first nibble received is always a code nibble. These observations are crucial as they simplify the logic of the decoder, and hence reduce the FPGA resources used. By subtracting 1 from the value of a code we obtain the number of data nibbles preceding the next code. Using a state machine, we loop through all the codes and extract the associated data. Once all the nibbles have been processed the *data_rcvd* port (see Figure 6.8) of the decoder is asserted and the state machine resets in anticipation of a new packet.

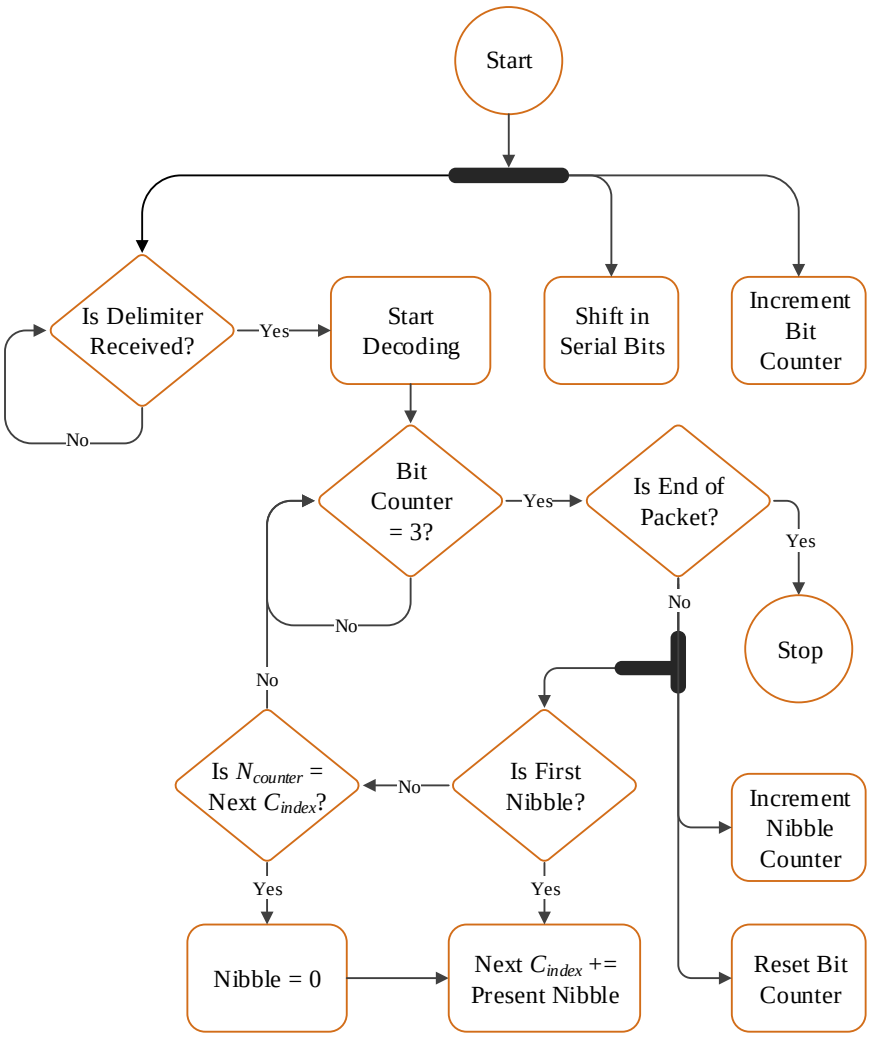


Figure 6.10: Flowchart describing the CONS decoder's implementation

6.4.4 Address-Inclusive Encoding and Decoding

A generic approach is taken in the implementation of the encoder so that it is usable for varying numbers of data width. In the vanilla implementation, where there is no addressing or any special bits inserted in the packet, the codec is intended for P2P communication without provision for addressing. However, the applications that require CELOC for use in a NoC would benefit from an addressable codec that has the addressing functionality embedded in it.

More often than not, the packet in Table 6.4 will need addressing if CELOC is used to implement an on-chip network. In appending the address bits to the packet, a plain un-encoded non-zero addressing is recommend, where a zero address is not used and the address bits therefore do not need to be encoded. This ensures the address bits do not eat into the maximum number of bits available for data. More important though, is the fact that in a CELOC-based NoC, the packet can therefore be routed through the network with much less latency since the intermediate nodes do not necessarily have to receive the entire packet as the address to deflect a packet to is visible in the packet. There is a *use_addr* port, with a corresponding address ports on the interface of the encoder and the decoder. This is used to enable the address-inclusive mode and is controlled by the TIL. Example waveforms showing the non-addressable and address-inclusive modes of the CONS encoding and decoding can be found in Appendix D.

6.4.5 Task Interface Logic

The *Task Interface Logic* (TIL) interfaces the task to the CONS Encoder and Decoder. It glues together a *Task Controller* (TC), Mutex, multiplexer (Mux), and Synchronizer (Synch). The Mutex is a means of sending status information out of the task and wrapper, especially for the purpose of error detection. For instance, the decoder could fail due to an error in its internal state machine's state transition. The Mux is used to choose between the output of the task and that of the Mutex.

The function of the TC is to start the task if it is idle and data has been received by the decoder. It also deserializes the received packet, recovers the data and presents it to the task. In addition, it handles addressing when the address-inclusive CONS is

used. These are the functions of the TC in this prototypic implementation. However, in an RC system that deploys CELOC, the TC would receive *command packets* from a system-level *Task Communication Manager* (TCM) to start or stop its associated task. It would also receive the destination address and the system time instance the processed data should be sent. This time however, cannot be earlier than the time instance the task finishes execution. If the packet is a *command packet*, the TC would check whether to start the task or reset it based on the command and would do accordingly. Otherwise, the received packet would be handled as a *data packet*. If the task is already started, the TC would route the new data to the task. The situation should not arise where a task is not ready for a new data, thus avoiding the need for buffering and saving on memory resources. This is because the TCM would dictate the time to send data based on when a destination task can accept it. It would therefore be counterintuitive to provide a buffering capability. However, a buffer can easily be inserted if necessary but the TCM's algorithm and the task computation model would have to be modified to account for this. In general, data should not be processed by the task at a rate faster than it can be routed through the CONS Encoder/Decoder (Codec) and the serial communication network except if buffering is used.

Similarly, in data delivery to the CONS Encoder is not buffered. The TIL ensures that the encoder is ready for a new input before applying the task's output data. The Synchronizer does this by checking that both *ap_ready* and CONS Encoder's *ready* are driven HIGH before asserting the CONS Encoder's *start*. It is guaranteed that once *ap_ready* goes HIGH, the data from the task is available as input to the CONS encoder. This is because the Output Data Mutex and Multiplexer are purely combinatorial and as such incur no clock delays. To ensure a non-buffered data at the input of the encoder, the encoding time should be accounted for in the timing model a system deploying CELOC.

6.4.6 Resource Utilization and Performance Evaluation

Table 6.5 shows the resource overhead of the network adapter for the bare SERDES and the packet-synchronized version. Tiny finite state machines are implemented for the PISO and SIPO blocks of the SERDES. These incur a total of 13 slices while the

CONS-based adapter's utilization amounts to only 32 slices. As the reconfiguration frame, which defines the minimum selectable area for partitioning the FPGA area, is always two columns (200 slices for CLBs). This implies that the adapter will fit right in, even with the smallest of tasks, by occupying only between 6.5% and 16% area of the reconfiguration frame.

While the use of the clock buffers for communication incurs a resource overhead of 32 slices owing to the encoding and decoding of serial data as expected in a serial transmission [229], this still offers a good area to performance ratio. Moreover, the big advantage of the buffers is that they avoid static interconnections between circuits. This facilitates a dynamic placement of circuits as will be shown in Section 6.6, and is fit for inter-task communication in reliability-aware applications as will be demonstrated in Chapter 7.

Table 6.5: Resource utilization of CELOC's network adapter

Resource	**Network Adapter Modules**				
	SERDES		**Packet-Synchronized**		
	PISO	***SIPO***	***CONS ENC***	***CONS DEC***	***CONS TIL***
FFs	30	72	65	100	84
LUTs	14	10	35	28	24
DSPs	0	0	0	0	0
BRAMs	0	0	0	0	0
Slice	13		32		

In terms of data transfer latency, the SERDES PISO block latches the parallel data in one clock cycle and streams it for the number of clock cycles equivalent to the number of bits in the parallel data. The SIPO recovers the data in the same period but uses one additional clock cycle for internal state transitions. As a result, the SERDES latency for a 32-bit data is 34 clock cycles.

For the evaluation of communication latency of the packet-synchronized adapter, the CONS encoder is directly interfaced to the CONS decoder and the number of cycles

measured individually for the encoder and decoder; and also for the entire codec. Table 6.6 presents the measured clock cycles. From the moment the encoder's *start* signal is asserted to the end of encoding and serial data shifting out (*ready* signal asserted), 56 clock cycles are incurred for the non-addressable codec and 60 for the address-inclusive version. Similarly, for the decoder, from the moment the FSS is received to the end of decoding, these numbers are 48 and 52 respectively. An entire 32-bit packet is sent and received in 60 and 64 clock cycles respectively.

For the different word sizes, the test packet is made a multiple of 32 bits for ease of comparison and the generation of the latency equations. That is, for the word lengths of 4, 8, 16, and 32, data widths of 32, 64, 128, and 256 bits are used. The equations are therefore, applicable only when the packet size is $8N_W$, where N_W is the word length in bits.

Table 6.6: CELOC CODEC's communication latencies for different word sizes

Word Size (N_W) in Bits	**Maximum Data Size Per Packet**	**Clock Cycle Latency for $8N_W$ Data Bits**					
		Non-Addressable			**4-bit Addressable**		
		ENC	***DEC***	***CODEC***	***ENC***	***DEC***	***CODEC***
4	56 bits	56	48	60	60	52	64
8	254 bytes	100	88	108	104	92	112
16	128 kB	188	168	204	192	172	208
32	16,384 MB	364	328	396	368	332	400
Latency Equation		$60 + 12(N_W - 4)$			$64 + 12(N_W - 4)$		

6.5 Clock Buffer Configurations for Network Access

The diagram in Figure 6.11 is a representation of a section of the Xilinx 7 series FPGA. The left and right clock regions are symmetrical with respect to the placement of clock buffers. Also indicated, are *Circuit Regions* (CRs) that are potential areas within the clock regions for task placement targeting the exploitation of clock buffers for communication. Note that not all possible CRs and communication routes are shown.

The notion here is that inter-circuit communication should only be through the clock buffers for the purpose of avoiding the static routes completely in order to aid the flexible relocation of circuits. Figure 6.11 also shows the locations of the clock buffers on the FPGA. In general, a 7 series FPGA (apart from the smaller Artix-7 chips) has 4 BUFRs, 2 BUFMRs, and 12 BUFHs in each clock region. In addition, there are 32 BUFGs at the centre of the chip, common to all the regions.

The clock buffers have different spans determining where communicating circuits can be placed. As stated in Section 6.1.2, BUFRs can drive clocking points in a single clock region, and thus can be used for intra-region communication. On the other hand, BUFMRs can drive resources in three vertical adjacent regions, providing a communication access that is 3-clock-region-wide. BUFHs are located in-between horizontal adjacent clock regions and thus can serve as communication links between them. Able to reach every clocking point, BUFGs are located at the centre of the FPGA and can be used for device-wide communication.

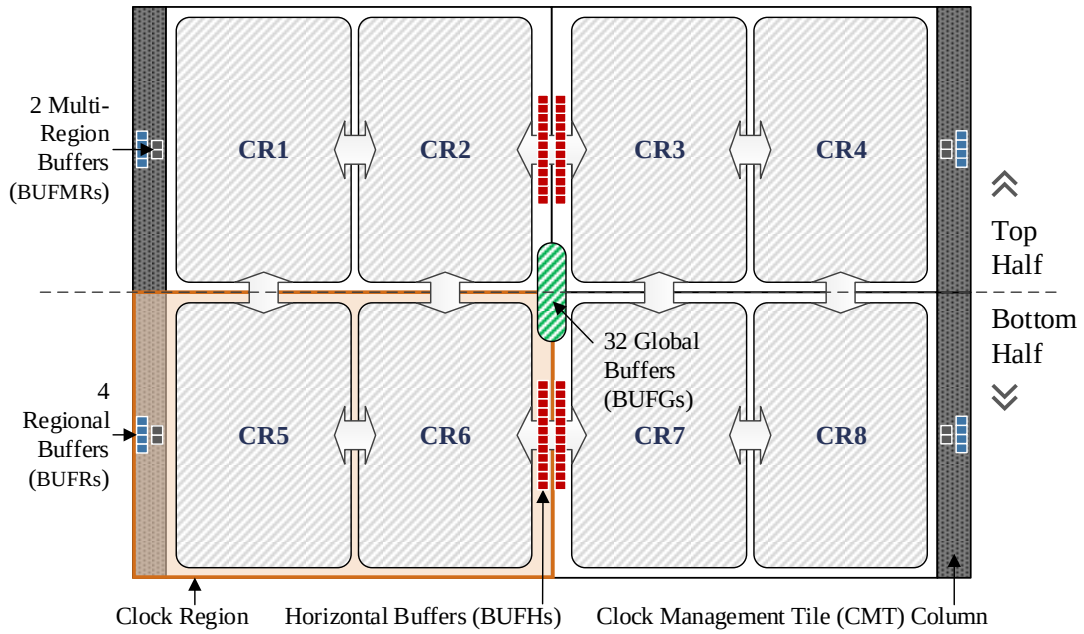


Figure 6.11: Representation of the 7 series FPGA chip, showing the locations of clock buffers, circuit regions (CR1 to CR8) for placing circuits within clock regions, and sample vertical and horizontal interconnections between the CRs

While the buffers can be used to communicate in different directions, certain configurations or combinations have to be used in order to provide communication.

This brings the question of data transfer speed since by connecting one buffer to another, a delay is introduced into the communication path. In the next subsections, we present some possible configurations of the clock buffers and their use cases. What informs these configurations are the reach of the buffers and the availability on them of clock enable (CE) pins that can be actively toggled. Furthermore, there is a restriction on buffer-to-buffer interconnections. For instance, it is only a BUFR that can feed a BUFG directly. As a result, if an intra-clock-region network has to access other clock regions not within its neighbourhood (immediate vertical and horizontal regions), a BUFR has to be used to drive a BUFG in order to feed such regions with data_clock. The following further points should be noted in the light of using different interconnections of the clock buffers for communication: All the buffers have physical CE pins; however, BUFMRs can only drive BUFRs, and both BUFHs and BUFGs can drive both BUFMRs and BUFRs.

Since only the BUFGs and BUFHs can drive the SR/PRE and CE inputs of registers, and it is essential for *data_clock* to be interfaced with a receiving circuit through an input that can be driven by a clock buffer, not all of the possible buffer combinations can be used if relocation support is sought. Only the ones that have data_clock coming out of a BUFG or BUFH can be used.

In the subsections that follow, it should be noted that arrows represent the direction of data transfer. For instance, with respect to Figure 6.11, CR1→CR8 means CR1 is transmitting to CR8 while CR2↔CR3 implies a bidirectional data transfer between CR2 and CR3. Moreover, since the clock regions have some symmetry, a communication like CR1↔CR3 is treated the same as CR2↔CR4; and CR1→CR6 as CR5→CR2. Also, except where indicated in the figures, all the connected clock buffers are in the same clock region, save for the BUFGs, which do not belong to any clock region.

6.5.1 Clock Buffer Configurations for Global Communication

Since the BUFGs have a device-wide reach, they can be used to transmit data to anywhere on the chip. This prevents any circuit region from being excommunicated from other regions. For instance, to communicate directly with CR-8 from CR-1 in

Figure 6.11, a BUFG has to be used (see Figure 6.12(a)). However, since the BUFGs are at the centre of the chip, an intermediate buffer has to be used to get access to them. There are four options – BUFR, BUFMR→ BUFR (used in Figure 6.12(a)), BUFH→ BUFR, or BUFH→ BUFMR→BUFR, depending on the available buffers and the relative location of the task. Depending on the specific application, either can be used.

In contrast, for a global communication like CR2↔CR7, there is a direct access to BUFGs. Therefore, the BUFG-only configuration shown in Figure 6.12(b) can be used. Note that any communication that does not reside within the same clock region, or that reaches only to an immediate vertical or horizontal region is classified as global.

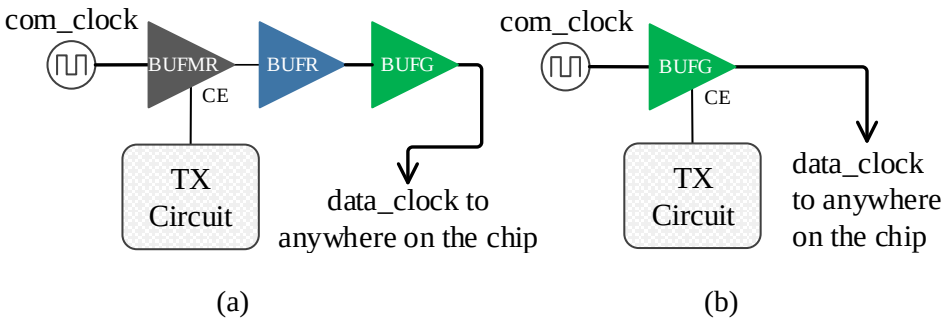


Figure 6.12: Clock buffer configurations for global communication showing (a) [BUFMR→BUFR→BUFG→], and (b) [BUFG→]

6.5.2 Clock Buffer Configurations for Horizontal Communication

By placing circuits at the inner edges of two horizontal adjacent clock regions, advantage can be taken of the BUFHs for horizontal communication. However, circuits at the outer edges require other configurations. With respect to Figure 6.11, and picking the top clock regions for illustration, the communications that fall into the category of horizontal inter-clock region include CR1→CR2, CR1→CR3, CR1→CR4, CR2→CR3, and CR2→CR4. Communications like CR2↔CR3 and CR2→CR1 can be achieved by using the configuration in Figure 6.13(a). However, since there is no clock-buffer-based physical link between CR1 and CR2 in the middle of the clock region, while CR2→CR1 can use [BUFH→], CR1→CR2 can use [BUFMR→BUFR→] (see Figure 6.13(b)). All the other communications have to use the configuration [BUFMR→ BUFR→BUFG→] presented in Figure 6.12(a). It should

be noted that CR2→CR1 can also use the configurations in Figure 6.13(c) and Figure 6.13(d).

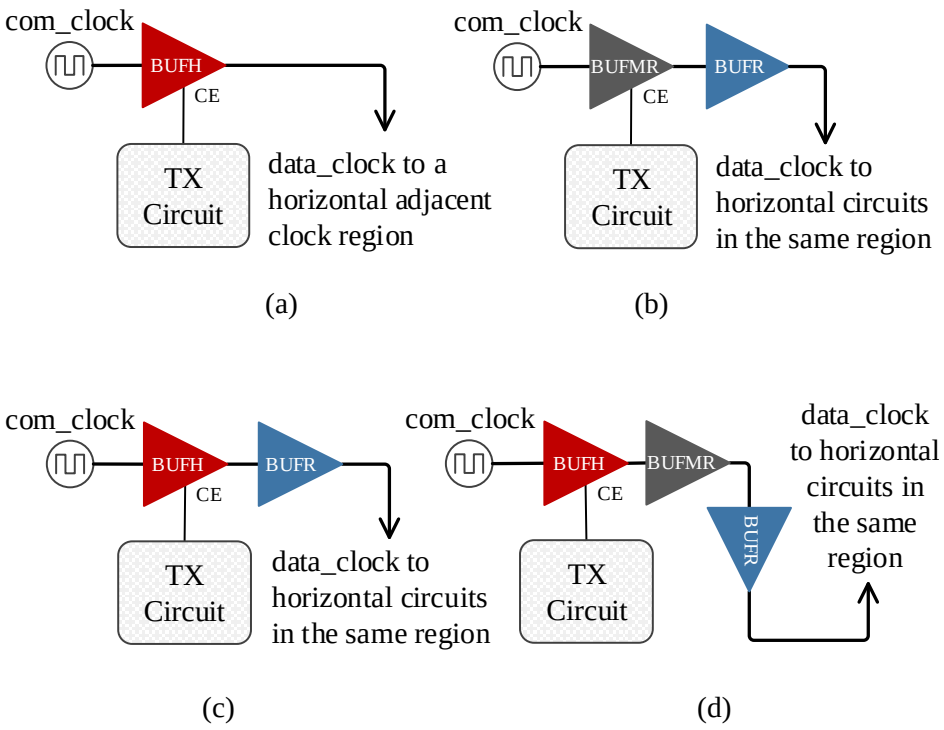


Figure 6.13: Clock buffer configurations for horizontal inter-region communication showing (a) [BUFH→], (b) [BUFMR→BUFR→], (c) [BUFH→ BUFR→], and (d) [BUFH→BUFMR→BUFR→]

6.5.3 Clock Buffer Configurations for Vertical Communication

From Figure 6.11, and picking the left clock regions for illustration, the communications that fall into the category of vertical inter-clock region include CR1↔CR5, CR1→CR6, CR2→CR5, and CR2↔CR6. Note that symmetry can be used to pick other ones. CR2→CR5, CR6→CR1 and CR2↔CR6 communications can be through the configuration in Figure 6.14(a) and [BUFG→] presented in Figure 6.12(b). All others can use the one in Figure 6.14(b). Note that the BUFRs in Figure 6.14 are not in the same clock region as the other buffers. As a result, they are indicated as BUFR(adj) in the figure caption.

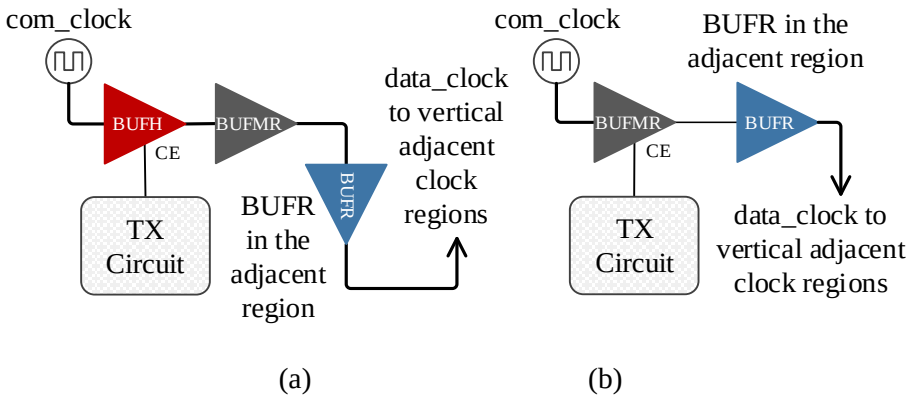


Figure 6.14: Clock buffer configurations for vertical inter-region communication showing (a) [BUFH→BUFMR→BUFR(adj)→], and (b) [BUFMR→ BUFR(adj)→]

6.5.4 Maximum Speeds of the Clock Buffers and Nets

The understanding of the maximum frequency of operation of the clock buffers is crucial in the attempt to adapt them for on-chip communication. This will help evaluate the limiting factor in the achievable communication speed of CELOC for different devices. Table 6.7 shows the maximum frequencies to which the clock buffers, nets, and trees can be driven. These numbers can be found in the respective datasheets for the Spartan-7 [231], Artix-7 [232], Kintex-7 [49], and Virtex-7 [233] FPGAs. At the time of writing, a Spartan-7 with speed grade -3 does not exist.

6.5.5 Bandwidth Characterization

To determine the maximum speed that each clock buffer configuration can achieve, the experimental setup in Figure 6.15 is used. The device used for the experiment is the Artix-7 (xc7a35tcpg236-1) FPGA with speed grade -1. The TX circuit is used to generate predetermined data packets to be received by the RX circuit. To confirm the correctness of data transfer, the validation of the received data is done by using an ILA to observe the signal transitions. A PLL clock generator is used to sweep the communication clock's frequency to the maximum value that still meets timing and does not corrupt the communication.

Table 6.7: Maximum operating frequencies of the clock buffers in the 7 series FPGAs

Device	Speed Grade	BUFG [Tree] (*MHz*)	BUFH [Buffer] (*MHz*)	BUFMR [Buffer] (*MHz*)	BUFR [Tree] (*MHz*)
Artix-7	-3	628	628	680	420
Kintex-7		741	741	800	600
Virtex-7		741	741	800	600
Spartan-7	-2	628	628	-	375
Artix-7		628 (394[a])	628 (394[a])	680 (600[a])	375 (315[a])
Kintex-7		710 (560[a])	710 (560[a])	800 (667[a])	540 (450[a])
Virtex-7		710	710	800	540
Spartan-7	-1	464	464	-	315
Artix-7		464	464	600	315
Kintex-7		625	625	710	450
Virtex-7		625	625	710	450

[a] At speed grade -2LE, *0.9 V*

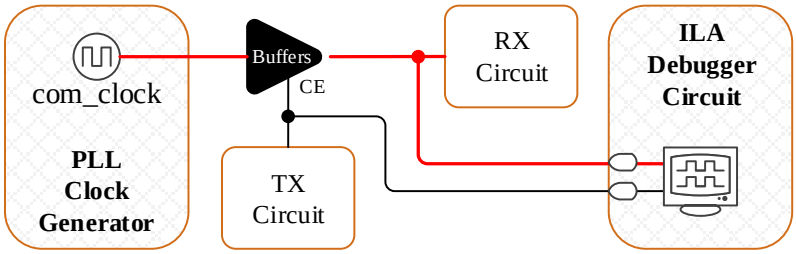


Figure 6.15: Experimental setup for characterizing the clock buffer configurations

The test communication packet is a 44-bit packet comprising of 32-bit data, a frame synchronization sequence of 8 bits and a serial encoder overhead of 4 bits. Table 6.8 shows the maximum bandwidths (data transfer rates) at which the buffer configurations work without a corrupted data transfer. The corresponding data recovery register or latch is also indicated. Moreover, based on the number of clock buffers in the FPGA, the number of instances of each configuration (not considering buffers used by other

configurations and those used for other purposes) that can coexist on a single chip is also indicated in Table 6.8.

Table 6.8: Bandwidth of the clock buffer configurations

Clock Buffer Configuration	**FDPE /LDPE**	**Bandwidth (Mbps)**	**Instances Per Clock Region**
BUFG→	FDPE	266.67	32 per chip
BUFR→BUFG→	LDPE	171.43	4
BUFMR→BUFR→BUFG→	LDPE	171.43	2
BUFH→BUFR→BUFG→	LDCE	187.50	4
BUFH→BUFMR→BUFR→BUFG→	LDCE	171.43	2
BUFH→BUFMR→BUFR(adj)→BUFG→	LDCE	171.43	2
BUFH→	FDPE	266.67	12
BUFMR→BUFR→	LDPE	240.00	2
BUFH→BUFR→	LDPE	275.00	4
BUFH→BUFMR→BUFR→	LDPE	240.00	2
BUFH→BUFMR→BUFR(adj)→	LDPE	240.00	2
BUFMR→BUFR(adj)→	LDPE	240.00	2
BUFR→	LDPE	233.33	4
Average	**-**	**221.15**	**-**

The highest speed of 275 Mbps is observed with the [BUFH→BUFR→] configuration and the lowest (171.43 Mbps) with the [BUFR → BUFG →] and [BUFMR → BUFR → BUFG→] configurations. Indeed, BUFR appears to be the limiting buffer. This is because it is the buffer with the slowest speed. In fact, for the Artix-7 board used for the experimentation, the speed grade is -1 and the BUFR's maximum frequency is graded at 315 MHz, which is the minimum for any of the buffers in the chip (see Table 6.7). Therefore, the average speed of 221.15 Mbps is relatively high considering that it is only 29.79% short of the BUFR's maximum. By extension to other speed grades, the average CELOC speed can be expected to scale as 70.21% of f_{BUFR_MAX}, where f_{BUFR_MAX} is the maximum frequency of the BUFR net in the target device.

In addition, the use of the clock buffers does not impact negatively on the number of circuits that can be on the FPGA simultaneously. The CRs can accommodate more than one circuit. Therefore, from the number of instances in Table 6.8, it is estimated that up to 16 circuits with communication access can be in a single clock region at the same time assuming they can be partitioned such that the CE access does not constitute a static route problem. This number is arrived at by excluding configuration instances that use similar buffers. In addition, there are 32 global configuration instances that can be shared by all the clock regions.

6.6 Dynamic Communication via Clock Nets

Partial Bitstream Relocation (PBR) is a key mechanism for enabling fault tolerance to permanent chip damages by relocating affected circuits to damage-free areas in runtime. In addition, PBR is fundamental to the management of tear and wear to delay the eventual occurrence of chip damages due to ageing (see Section 3.2.3). Meanwhile, PBR is not supported by vendor tools and PR flow, thereby necessitating that all routes between inter-communicating circuits are statically determined at compile time. For full relocatability, a dynamic communication infrastructure is needed.

This section advances a dynamic communication access mechanism that is termed *Clock-Enabled Relocation-Aware Network-on-Chip* (CERANoC). This builds on CELOC with the main advantage for CERANoC being that the clock buffers and nets use dedicated routing wires that are independent of the general logic interconnect. This removes the restriction of the static interconnect links and enhances the online relocation of circuits. This mechanism relies on the replacement of the interconnect links in NoCs with clock buffers. Since the clock buffers do not use the general logic routing resources, the path from a transmitting circuit to a receiving circuit is free of general logic interconnections.

As discussed in 3.2.5, for circuit relocation to be feasible, communication must be provided for the circuit being moved at the resource-matching destination. With regards to Figure 6.16, the easiest way to provide this communication is to ensure that a route

from Task 1 to LOC 2 is established at design time. This way, during runtime, Task 2 can be moved to LOC 2 while maintaining its communication link with Task 1.

The solution to dynamic communication in CERANoC eliminates the static inter-circuit communication routes all together. Using Figure 6.16 as an example, this is achieved by removing the static inter-task connections and replacing them with clock buffers as shown in Figure 6.17. The hypothetical layout of tasks here is the same as that in Figure 6.16, except that the interfaces between Tasks 1 & 2, and between Tasks 1 & 3 have been removed. To provide communication, a clock buffer is used to transmit serial bits from Tasks 1 to 2 and 3. This signal also feeds LOC 2, so that if Task 2 is relocated to LOC 2, the communication between it and Task 1 remains intact. At the same time, LOC 1 is now free of a crossing routing. Basically, the surface of the chip is generally freed of inter-circuit routings.

In the implementation of CERANoC in this work, the clock regions of the FPGA are used as NoC nodes. The clock buffers are pre-routed between clock regions at design time so that during runtime, regardless of the clock region a task is placed, it is able to communicate with any task in any other clock region. The main concept behind the technique is to allow serial data to ride on a clock signal from a source node to a destination node.

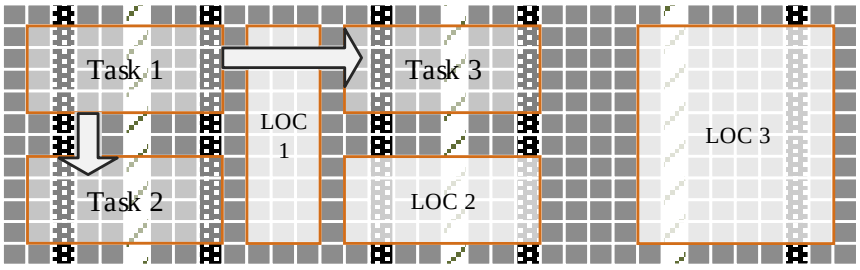


Figure 6.16: Diagram demonstrating how static routes hinder relocation. Task 2 cannot be moved to LOC 3 without preserving the existing interconnections

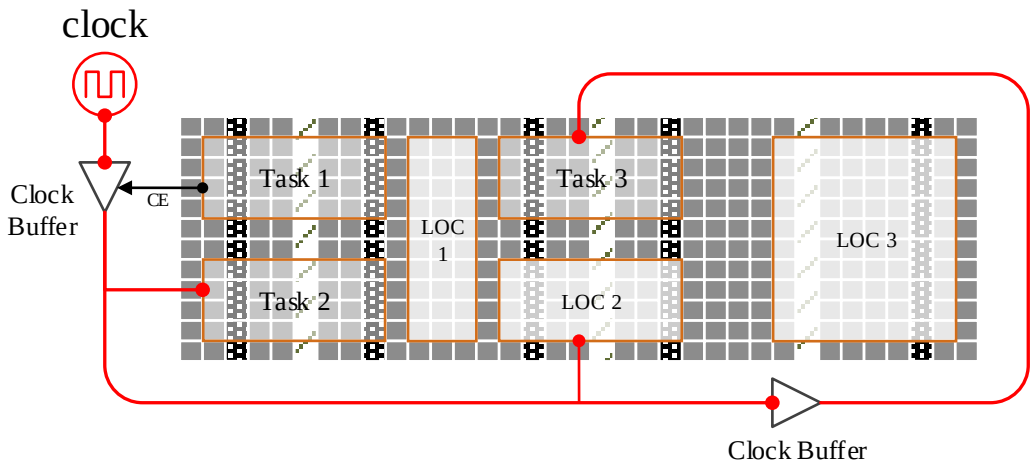


Figure 6.17: By removing the inter-circuit interfaces and replacing them with clock buffers, it is possible to achieve dynamic communication

Shown in Figure 6.18 is the classic mesh topology for CERANoC. With respect to the clock buffer configurations, vertical node-to-node interconnections are achieved with [BUFMR→ BUFR(adj)→] and horizontal communication with [BUFH→]. Node-to-node bandwidth in this topology would be expected to be an average of 237 Mbps.

Other topologies can be formed easily by using a relevant combination of the clock buffer configurations presented in Section 6.5. The diagram only shows four nodes, but this can be easily extended as the dotted lines depict. Clock buffers are connected in-between nodes as shown in Figure 6.18 and Figure 6.19. The clock buffer connections are based on the relevant configurations in Section 6.5. In the following subsections, the special considerations for the implementation of CERANoC are presented in relation to the design parameters of a traditional NoC. It should be noted that the aim is to implement a full-fledge NoC as that is beyond the scope of this work; rather, the intention is to demonstrate how the use of clock buffers for inter-circuit communication enhances the relocatability of hardware tasks.

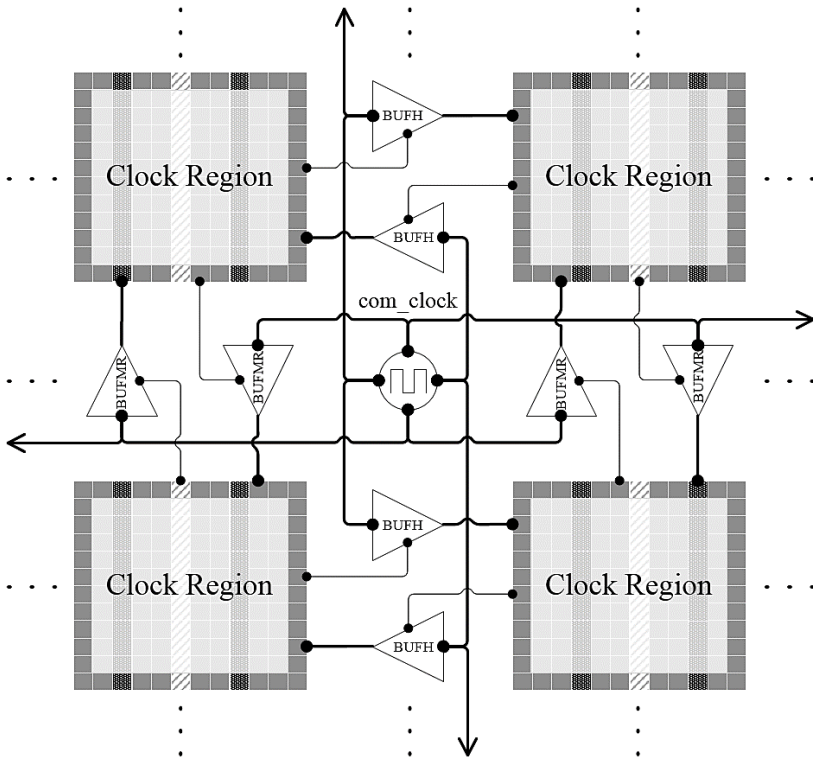


Figure 6.18: 4-node CERANoC mesh network showing inter-clock region connections achieved with clock buffers

6.6.1 Packet Format and Addressing Scheme

The FSS, the destination address, the CONS overhead, and a data of 56 bits maximum are all concatenated into a single packet as shown in Table 6.9. A unique *Node ID* is given to each node on the network. This ID also serves as the address of the node and is added to the communication packet as the destination address. With *N* nodes, the address range is from 1 to *N*, with zero deliberately avoided because the address field in the packet is un-encoded as noted for the address-inclusive packet encoding and decoding in Section 6.4.4.

As such, one reason for transmitting the address in plain format is that the routers need to know the destination address before routing the packet. Encoding and decoding the address would incur further clock cycles at the encoder and the decoder. In addition, this would require more logic resources for the decoding and re-encoding functions in the router. Furthermore, an encoded address would eat into the number of bits available for the actual data, eventually preventing data transfer as *N* increases and becomes 56.

Therefore, with this careful choice of an address range, CERANoC saves on time and resources, and ensures a maximum data transfer throughput.

Table 6.9: Packet format for 4-bit-data-word CERANoC

Fields	***FSS***	***Destination Address***	***CONS Overhead***	***Data***
Number of Bits	8 bits	*N* bits[a]	4 bits	56 bits max
Description	Value: 80 hex	Unencoded	CONS-encoded	

[a] *N* = number of nodes

6.6.2 Network Routing

A routing algorithm determines the routing of data from the source to the destination in a network. The problem of designing routing algorithms that meet different performance and architectural requirements has been extensively studied. Some of these requirements are low latency, low power consumption, scalability, and programmability [176]. CERANoC supports any existing routing algorithm so far the clock buffers can be arranged to serve as links in the topology chosen. There is no other special consideration for routing in CERANoC. For instance, a Torus CERANoC can use BUFGs to connect the topmost clock regions to the tail regions and the leftmost regions to the rightmost regions.

6.6.3 Prototype Network Demonstration

To demonstrate the feasibility of CERANoC, a 4-node prototype star network with a *Central Router* is implemented on the Artix-7 (XC7A35TCPG236) FPGA chip (see Figure 6.19). Figure 6.20 shows the global clock generation and distribution architecture. A special *switch_clock* is needed by the *Central Router*. This is from the same output that feeds the BUFG which distributes com_clock. The clock buffer configuration used by the nodes is [BUFMR→BUFR→BUFG→] but could have also been [BUFR→BUFG→]. Note how the BUFGs are located logically inside the *Central Router*. Because *switch_clock* feeds these BUFGs, passing it through another BUFG would adversely affect network bandwidth. Moreover, by directly feeding the BUFGs, *switch_clock* ensures that on the part of a receiving node, a received packet arrives

directly from another node with the data clock having been refreshed. This is because as the *data_clock* from the BUFR, having gone through a BUFG, enters the *Switch Arbiter*, it is received by an LDPE latch and passed through a crossbar logic before being fed to the CE of a BUFG, essentially starting a new transmission (see Figure 6.21).

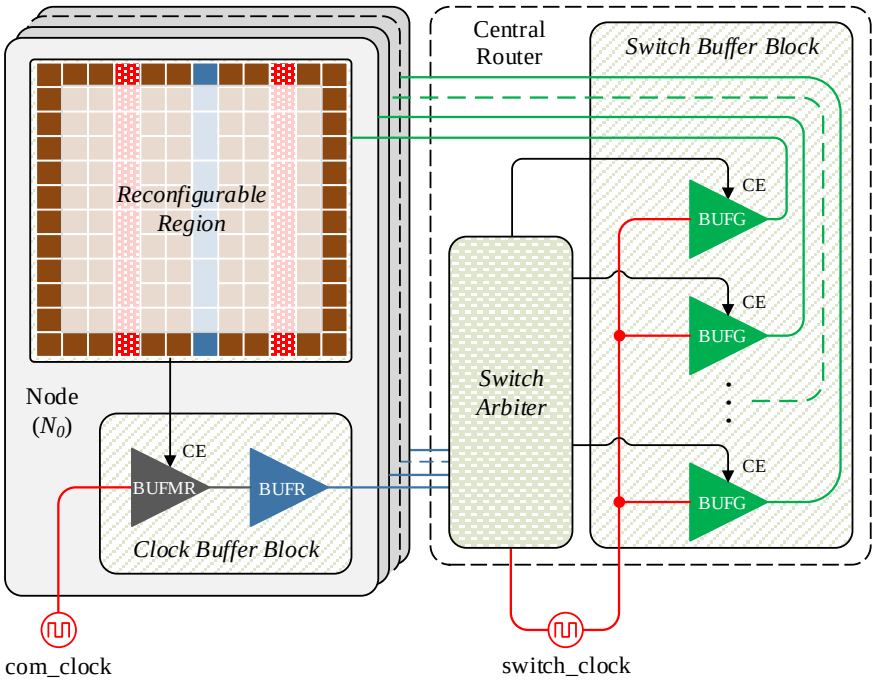


Figure 6.19: 4-node CERANoC star network using clock buffers as network links

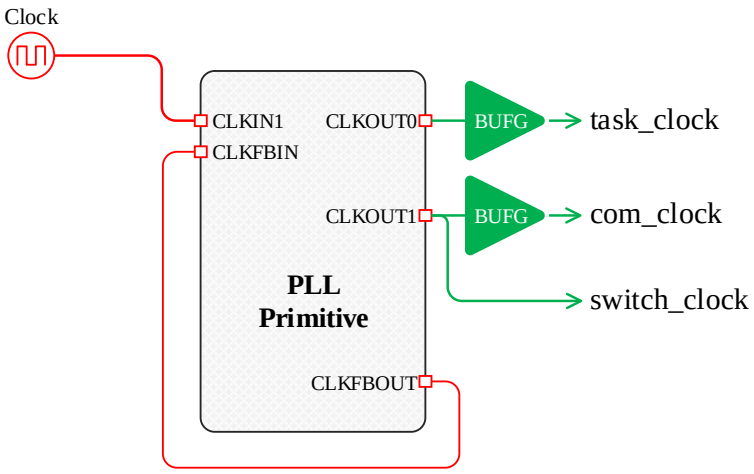


Figure 6.20: PLL-based global task and communication clock generation and distribution

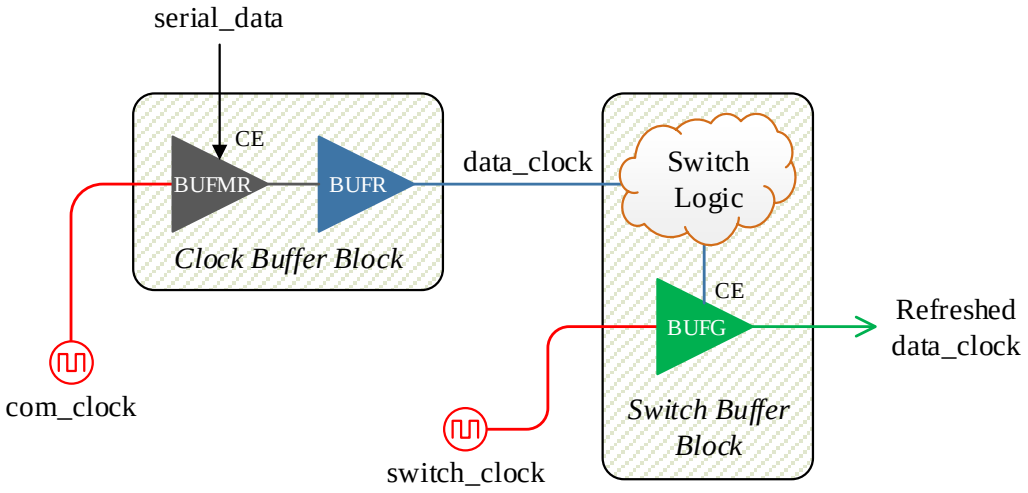


Figure 6.21: Data clock renewal as it traverses the centre of a star-shaped CERANoC

Since the objective of this prototype demonstration is to show that using the clock buffers in the manner stipulated by CELOC/CERANoC facilitates dynamic communication and circuit relocation by circumventing the general interconnect, most of the intricacies of NoC designs are avoided as these are already extensively studied. A point-to-point routing is used for the star-network (see Figure 6.22), and a 32-bit payload data is used, giving a 48-bit packet. 48-bit buffer memories (48 x 1-bit LUT-RAMs) are provided inside the *Central Router* in order to temporarily store packets that cannot be immediately routed. In order to control access to multiple nodes attempting to transfer packets simultaneously to the same receiving node and thus keep in line with real-time requirements, priorities are assigned to the nodes based on the node address. A node with a lower address has a higher priority.

Figure 6.22 shows the switch architecture implemented for the 4-node star network. N_i implies node i while S_{jk} denotes a switch position from node j to node k. The indicated positions of the switches are for the following routing: $N_0 \rightarrow N_3$, $N_1 \rightarrow N_0$, and $N_3 \rightarrow N_1$. There is one *Node Router* (NR) for each input to the Switch Arbiter. In each NR there are three $(N - 1)$ independent switch endpoints (S_{jk}) which determine the routing of the incoming packet to the other three nodes. In the Switch Arbiter, there is a 4-bit *occupied_switches* register that shows the state of the nodes with respect to data reception. A node that is presently receiving a packet has its corresponding bit turned on. The Switch Arbiter checks the destination address of the packet against the

state of the occupied switches. If the destination node is not already occupied by an ongoing transmission, the packet is routed through and the *occupied_switches* state is updated.

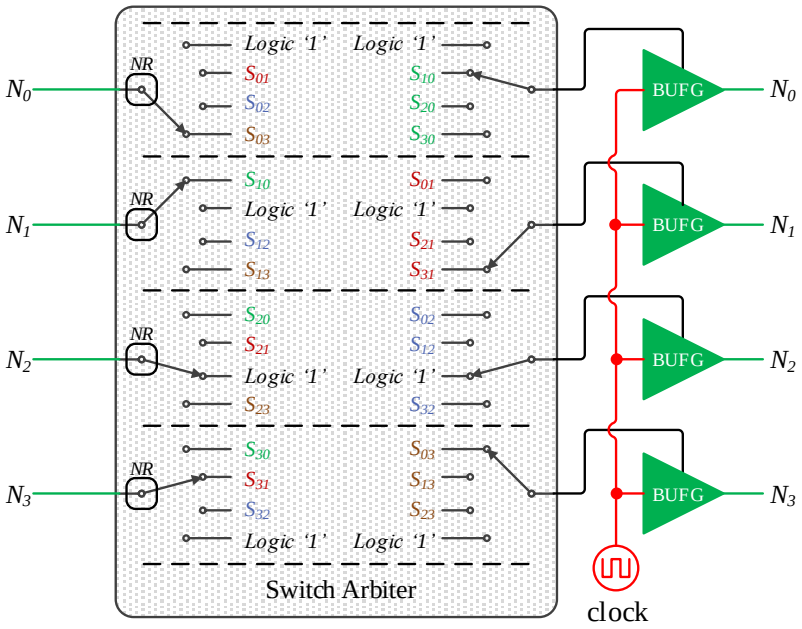


Figure 6.22: Switch architecture for a 4-node CERANoC star network

To test dynamic communication and relocation at the fundamental level, four tasks (θ_0 to θ_3) are set up, with one task in each of the nodes. It is very important in this demonstration to have a visual indication that new tasks are able to establish communication and that existing tasks still execute correctly when new tasks are placed in runtime. As a result, a VGA application is used with the setup shown in Figure 6.23.

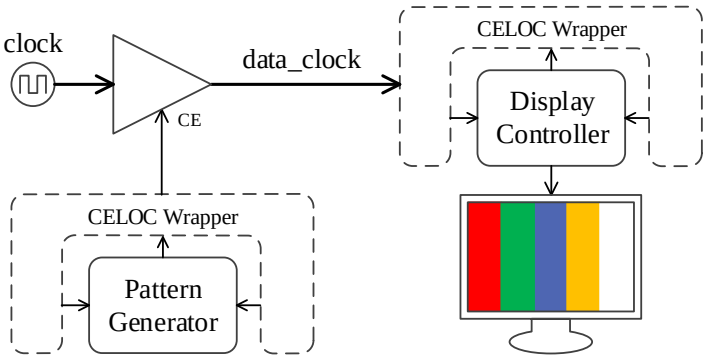


Figure 6.23: Setup for demonstrating CERANoC

Tasks θ_0 to θ_2 are pattern generators, each generating three different patterns (P) of four vertical stripes of colours white (W), red (R), green (G), or blue (B). Each of these coloured stripes is represented by 8 bits (3 bits for R, 3 bits for G, and 2 bits for B). Every 32 bits of data sent by a pattern generator, therefore, determines four stripes of 8-bit colour. θ_0 to θ_2 generate P_0 to P_2 respectively, with P_0 = [W, R, G, B], P_1 = [G, B, W, R], and P_2 = [B, W, R, G]. θ_3 is a fixed VGA controller that interfaces to a VGA monitor in order to display the patterns generated by θ_0 to θ_2. At design time θ_0 to θ_3 are floor-planned in nodes N_0 to N_3 respectively and partial bitstreams are generated for only θ_0 to θ_2. Task θ_3 has to be static because it needs access to the VGA's interface pins which are in fixed locations on the FPGA. Tasks θ_0 to θ_2 are set to transmit to θ_3 at the same time. Because of the router priority, this means P_0 is continuously displayed. By blanking N_0 and N_1 successively using blanking bitstreams, we are able to see P_1 and P_2; reconfiguring N_1 then N_0 also results in patterns P_1 then P_0, demonstrating that communication in the network is unimpaired when tasks are swapped in and out in runtime.

The demonstration of relocation involves configuring θ_0 in N_1 while blanking N_0 and θ_1 in N_0 while blanking N_1, though, after necessary bitstream manipulations to change the target frame address. In the former case, we are able to see pattern P_0 even though it is configured in N_1, and vice-versa for the latter case. The *Vivado Hardware Manager* is used to configure the partial bitstreams. Figure 6.24 shows the floorplan of the FPGA after implementation. It can be seen that the chip areas belonging to nodes 0 to 2 are free of general routing. This is as expected. The Network Interface does not contribute to static routing as it is made part of the reconfigurable task itself. Only the clock lines can be seen routed in the HROW from a global network feeding the clock regions (refer to Figure 6.1 in Section 6.1.1). These routings are dedicated clock nets and do not interfere with relocation. This means the clock regions remain free of general routing even though they are interconnected. The connections to CEs are at the edges of the clock regions, leaving the majority of the region free of general routing.

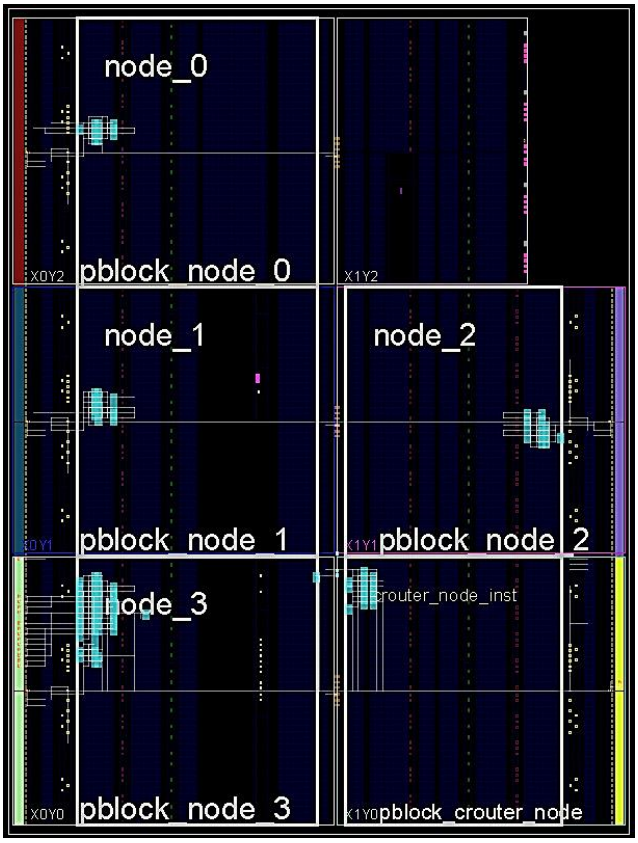


Figure 6.24: Floorplan of the implemented 4-node network

A. Resource Utilization

The only component peculiar to CERANoC is the network adapter and it uses only 32 slices (see Section 6.4.6). The entire 4-node network itself (without the tasks) takes 144 slices. Clock buffer utilization stands at 4 BUFMRs, 4 BUFRs, and 6 BUFGs, with per clock region utilization of 50%, 25%, and 3.125% respectively.

B. Network Latency

Since the central router simply routes the packet from the source to the destination nodes, essentially effecting the connection between [BUFMR→ BUFR→] and [BUFG→] in a [BUFMR→BUFR→BUFG→] clock buffer configuration, the packet transfer latency remains 64 clock cycles as presented in Table 6.6 in Section 6.4.6 for packet-synchronized address-inclusive encoding.

C. Network Throughput

The CONS encoder and decoder do not share circuitry. Therefore, nothing stops concurrent data transfers like these four simultaneous data transfers: $N_0 \rightarrow N_1$, $N_1 \rightarrow N_2$, $N_2 \rightarrow N_3$, and $N_3 \rightarrow N_0$. That is, for the 4-node star CERANoC, the throughput of the individual link can be multiplied by 4 to obtain the network throughput. As such, for an *N*-node star CERANoC in full-duplex mode, the throughput (in Mbps) can be defined by Equation (6.1) in terms of the payload size (in bits), the number of nodes (*N*), the frequency of operation (*f* in MHz), and the latency cycles as follows:

$$Throughput\ (in\ Mbps) = \frac{Payload\ (in\ bits) \times N \times f(in\ MHz)}{Latency\ Cycles} \tag{6.1}$$

At 100 MHz this gives a throughput (data rate) of 200 Mbps for the network demonstrated. The CELOC links used has a maximum speed of 171.43 Mbps (same as 171.43 MHz since one bit is transmitted in one clock cycle). The maximum throughput for the Artix-7 device used is therefore, 428.58 Mbps for a 32-bit payload and N = 5 (assuming the *Central Router*'s RP is also used to host a node). It should be noted that the latencies for payloads other than 32 bits can easily be determined from Table 6.6.

Compared with methods that involve runtime routing, CERANoC does not incur any clock cycle overhead in order to place a new circuit or relocate one in runtime. Moreover, compared with the method in [90], the ICAP is not required for communication purposes, thus allowing SEM to have as much ICAP time as possible. The use of the ICAP for communication could be counterintuitive where reliability is important. Moreover, while DyNoC [182] also achieves dynamic communication for newly placed tasks, it is not certain that it is able to support relocation since the problem of general routing seemed not to have been addressed. CERANoC on the other hand leaves the chip area clear of general routing.

6.7 Fault-Tolerant Data Transfer

The occurrence of soft errors in a communication network could manifest itself in the form of erroneous data transfers. For CERANoC, bit flips can occur in the flip-flops of

the network adapter leading to erroneous encoding or decoding of the received data. It is also possible for permanent damages to occur, for instance due to chip ageing or electro-migration, and in this case, the HEM methods discussed in Section 3.2.3 can be applied. For instance, the core and its associated network adapter can be relocated to a damage-free region by using a fault-tolerant configuration engine like the one in Chapter 4. In the following subsections, the modifications that have to be made to CERANoC in order to make it resilient to errors are highlighted.

6.7.1 Network Adapter

The diagram in Figure 6.25 shows a TMR design of the network adapter. Only the key blocks and signals are shown. The adapter in Figure 6.8 has been modified by inserting an error control (EC) decoder and an EC encoder in the path of the CONS decoder and encoder respectively. The resulting circuit is triplicated and a voting circuit used to monitor and compare the outputs of the TMR modules. If at least two of the outputs are the same, communication can continue and the offending adapter can be reconfigured. The EC block can be for CRC or SEC-DED, depending on the requirement of the application. CRC is only able to detect errors but not correct them. On the other hand, as a forward error correction mechanism, SEC-DED is able to detect and correct single-bit errors but only detect double-bit errors.

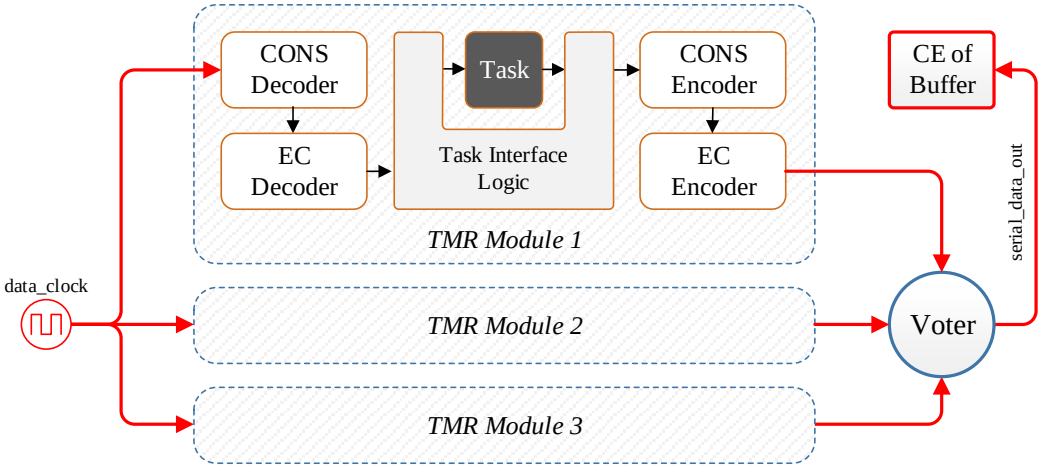


Figure 6.25: Fault-tolerant network adapter for CERANoC

6.7.2 Communication Packet Format

The structure of the packet for both the CRC and the SEC-DED error control implementations are shown in Table 6.10 and Table 6.11 respectively. For the CRC implementation, the FSS, the destination address, the CONS overhead, a data of 32-bit, and a 4-bit CRC Frame Check Sequence (FCS) are all concatenated into a single packet as shown in Table 6.10. A similar thing is done for the SEC-DED implementation except for the fact that the SEC-DED Error Control Code (ECC) bits are interleaved with the 32-bit data.

Table 6.10: Packet format for CRC-based error control

Fields	**FSS**	**Destination Address**	**CONS Overhead**	**Data**	**CRC FCS**
Num. of Bits	8 bits	*N* bits [a]	4 bits	32 bits	4 bits
Description	80 hex	Un-encoded	Encoded		

[a] *N* = number of nodes

Table 6.11: Packet format for SEC-DED-based error control

Fields	**FSS**	**Destination Address**	**CONS Overhead**	**32-Bit Data Interleaved with 7-Bit ECC**
Num. of Bits	8 bits	*N* bits [a]	4 bits	39 bits
Description	80 hex	Un-encoded	Encoded	

[a] *N* = number of nodes

6.7.3 Packet Error Control Implementation

For the CRC detection of erroneous transfer of data, a 4-bit checksum FCS with a Hamming Distance of 2 and a reverse-of-reciprocal polynomial 0x9 [60] is used. This means all 1-bit and some 2-bit errors can be detected. To minimize the impact on data transfer latency, a parallel CRC implementation is used. The 4-bit FCS is calculated on the data to be sent and appended to it. At the receiver, CRC is again calculated on the data and the result checked against the FCS contained in the received packet to verify the correctness of the received data. When a CRC error is detected, the packet is dropped and a resend request is sent to the source node. Additional data bit fields in the packet are used for this.

For the forward error correction scheme based on SEC-DED, the SEC-DED Hamming code (39, 32) is used. This code uses 7 parity bits for correcting single-bit errors, and detecting double errors in 32 data bits. Although, Figure 6.25 shows the EC Decoder and Encoder as being distinct blocks in the network adapter, in the actual implementation, they are absorbed into the CONS decoder and encoder respectively.

6.7.4 Pipeline Mechanism for Packet Transfer

The CRC-4 module adds only a 4 clock-cycle latency to the serial stream and does not impact the CONS encoding process. It is not impossible for a CRC calculation to give an FCS of zero. Since a zero cannot be part of the packet, to avoid ambiguity, the FCS itself has to be encoded by the CONS encoder. In addition, the CONS encoding uses a look-ahead algorithm, where a nibble early-on in the packet cannot be encoded until a zero nibble is found later on in the packet. If the CRC encoding were to be applied to the CONS-encoded nibbles, a latency of as much as the number of nibbles in the packet could be incurred. As a result, the CRC has to be applied to the data before CONS encoding is done. However, this does not delay the CONS encoding. Since the CRC is a non-intrusive error detection mechanism, that is, it does not modify the data, the CONS encoder can be pipelined to start at the same time as the CRC encoder. By the time the CONS encoder is on the last nibble, the FCS is ready. At the receiver, the CRC can only be applied after at least the first nibble in the packet is decoded. The 4 bits of a nibble are shifted in and decoded in 4 clock cycles. This means an unavoidable minimum of 4 clock cycle wait for the CRC decoder.

For the SEC-DED implementation, it is impossible to avoid waiting for the entire packet to be CONS-encoded before applying the SEC-DED algorithm. The SEC-DED is intrusive since the parity bits are interleaved with the data. Therefore, the pipelining of the CONS encoding and the SEC-DED encoding is not possible. However, to limit impact on latency, a completely combinatorial implementation is used for the SEC-DED algorithm, with no input registering or internal pipelining. Moreover, the implementation is such that the SEC-DED decoder corrects a single-bit error automatically and flags double-bit errors.

6.7.5 Evaluation of the Fault-Tolerant Network Adapter

Table 6.12 shows the resource and time overheads of introducing the CRC and SEC-DED error control into the CONS encoder and decoder. The data presented here is for a single instance of the TMR modules and is based on the implementation on an xc7a35tcpg236 FPGA chip. Only the relevant blocks (CONS encoder and decoder) are presented in the table. Without any error control mechanism applied, the CONS encoder and decoder require totals of 147 flip-flops and 88 LUTs, with the latency from the point of CONS encoding to the point of complete data reception being 60 clock cycles.

For the CRC-based error control, the utilization goes up to 157 (6.80% increase) for flip-flops and 109 (23.86% increase) for LUTs, with the total latency 68 clock cycles for an address-inclusive CONS codec. Compared to when no error control is applied, only 4 clock cycles have been added, arising from the transmission and the reception of the 4-bit FCS. Note that the latency measurements were made using a continuous data transfer and reception. This is implies that though, 8 clock cycles are expected for the transmission and reception of the FCS, only 4 are observed because of the overlap between transmission and reception.

On the other hand, the SEC-DED implementation uses the same number of flip-flops for the CONS encoder and decoder. This is because of the LUT-only implementation used. However, LUT utilization has surged to 199 (126.14% increase). The latency also increase by 9 clock cycles accounting for the time taken to transmit the added 7 bits and one clock cycle transitioning period each for the encoded and decoding of data respectively.

The implication of this result is that depending on the nature of the environment into which the network is deployed, either the CRC or the SEC-DED-based error control mechanism can find application. For instance, if the network is anticipated to experience several bit flips, then a forward error control mechanism would be better. Though the SEC-DED is more expensive in terms of hardware resources and latency, the cost of retransmissions that would be incurred in a CRC-based error control could easily offset this overhead in latency costs. On the other hand, if the system would be operating under a mild error-prone environment, then a CRC-based error control

implementation would suffice, as there would only be intermittent requests for retransmissions, which at the system level would be negligible.

Table 6.12: Logic resource overhead of the fault-tolerant network adapter

Error Control	**Module**	**Resource Utilization**		**Latency (cycles)**
		Flip-Flops	***LUTs***	
None	CONS Encoder	59	52	64
	CONS Decoder	88	36	
	Total	**147**	**88**	
CRC	CONS Encoder	64	71	68
	CONS Decoder	93	38	
	Total	**157**	**109**	
SEC-DED	CONS Encoder	59	80	73
	CONS Decoder	88	119	
	Total	**147**	**199**	

6.8 Chapter Summary

This chapter has presented a unique and novel adaptation of clock buffers as serial network links for on-chip inter-circuit communication. This has proven to not only reduce the utilization of general routing resources as network links, it has also been demonstrated to help avoid the traditional static routes that are bottlenecks to runtime partial bitstream relocation. To access a clock buffer for communication, a special adapter is wrapped around a task. This adapter utilizes only 32 slices and different configurations of clock buffers have shown an average bandwidth of 221.15 Mbps for an Artix-7 speed grade -1 device. A unique clock-buffer-based network access mechanism was also advanced and shown to facilitate dynamic on-chip communication for the purpose of circuit relocation. Finally, this chapter presented a fault-tolerant implementation of the network adapter to ensure error-free data packet transmissions.

Chapter 7

7

A Case Study of the NASA/JPL Compositional Infrared Imaging Spectrometer

In order to demonstrate the practicality of the proposed frameworks, a case study application is drawn from the NASA/JPL *Compositional InfraRed Imaging Spectrometer* (CIRIS). This spectrometer has data processing stages that can be used as hardware tasks. A system setup that involves the use of the configuration memory access manager (CAM) proposed in Chapter 4 and the inter-task communication framework advanced in Chapter 6 are used for evaluations. The latency and the extent of relocation supported are evaluated for three approaches to on-chip communication, with the target device chosen as the Zynq-7000 (xc7z100ffg900-2), which contains a Kintex-7 programmable fabric and a dual-core Arm processor. The CIRIS data processing RTL codes used for evaluation were provided by one of the investigators who designed an SoC-based controller for the CIRIS instrument [210].

Part of the investigations reported in this chapter has been covered in the author's publication in [234]:

- **A. Adetomi**, G. Enemali, Xabier Iturbe, Didier Keymeulen, and T. Arslan, 'R3TOS-Based Integrated Modular Space Avionics for On-Board Real-Time Data Processing', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 1–8.

7.1 An Overview of the CIRIS Spectrometer

The CIRIS is one of the new-generation NASA instruments proposed to search for life indicators in Jupiter's moon, Europa [208]. This moon orbits inside Jupiter's high-

radiation belts, and hence any mission sent there must withstand extremely harsh radiation. Therefore, the CIRIS avionics must be designed to have a high system-level reliability with emphasis on tolerance to high radiation doses.

CIRIS is a small, rugged and lightweight *Fourier Transform Spectrometer* (FTS) with a high *Signal-to-Noise Ratio* (SNR) in the near-IR to thermal-IR region (2-12 µm), where the strongest and most diagnostic vibrational bands of the compounds of interest in Europa are found (e.g., 'CHNOPS' functional groups). The major structural novelty introduced by CIRIS is its constant speed rotating refractor to vary the optical path difference of the two rays in which incoming light is divided at the entrance of the instrument using a beam splitter (red and green rays in Figure 7.1). The reflected rays in the rotating refractor recombine after travelling through the instrument, resulting in a fringe interference light pattern (interferogram) that is measured with a photo-detector (purple ray in Figure 7.1). Based on Snell's law, the light rays travel the same distance through CIRIS optics when they are incident on the rotating refractor at 45º. The Zero Path Difference (ZPD) positions occur when the refractor is parallel or perpendicular to the beam splitter, that is, four times over the course of a revolution. The regions where the optical interference between the input light rays can be measured are located at approximately 16º arcs around each of the four ZPD positions. As the CIRIS refractor performs 6.5 revolutions per second, the interferograms span over a period of 13.6 ms every 24.8 ms.

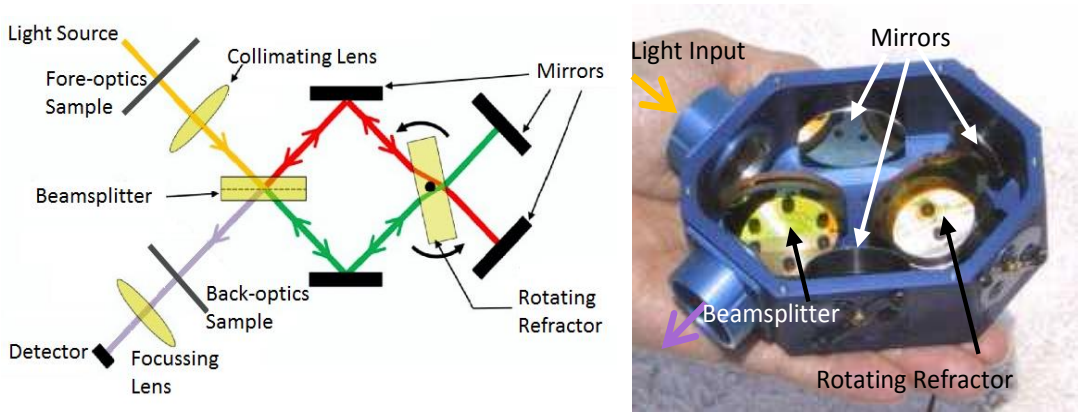


Figure 7.1: CIRIS interferometer [207]

7.2 CIRIS Data Processing

The CIRIS interferogram signal is conditioned, filtered and amplified to ±5V range prior to being digitized at 1 MSPS using an 18-bit resolution ADC. The interferogram samples are then processed to produce a spectrum that illustrates the intensity of the wavelengths present in the light beam. This in turn permits to find out the chemical composition of the sample under study by looking at the absorption lines in the spectrum. However, spectral leakage (i.e., "picket-fence" effect) and noise are also present in the spectrum due to the limited discretization of the interferograms through time limited digital sampling, and need to be properly processed by the instrument avionics to produce meaningful results [235]. The CIRIS data processing is accomplished in four stages.

The first stage prepares the interferogram for subsequent processing by selecting 8,192 samples centred on the ZPD position. This is done to deal with any temporal shift that might have occurred while sampling the interferogram. This stage also removes the DC offset in the ZPD-aligned interferogram by subtracting its average value, which is computed using a *Cumulative Moving Average* (CMA).

The second stage (STAT Inter.) computes the variance and performs a CMA on successive interferograms detected around the same ZPD positions with the objective of estimating and increasing the SNR by removing the effect of high frequency and random noise. The mean and variance results for every interferogram position after each algorithm iteration are stored in dedicated DDR memory segments.

The third stage apodizes the averaged interferograms at the edges of the sampled regions to minimize the effects of spectral leakage and computes the FFT on the interferogram. In the light of increasing spectral resolution, this stage adds 4,096 zeros to each of the tails of the interferogram to obtain 8,192 additional interpolated spectrum points in-between the original nonzero-filled spectrum data, that is, 16,384 total spectrum points. In order to obtain the best performance, the FFT stage pipelines several Radix-2 butterfly processing engines, where each engine has its own memory banks to store input and intermediate data. This pipelined implementation allows computing one spectrum data per clock cycle, with a latency of 33,013 clock cycles. A CORDIC logic

is then used to translate the real and imaginary representation of the spectrum data used in the internal butterfly processing engines into polar representation, which is more suitable for scientific analysis. This translation is completed within 36 clock cycles.

The fourth and last stage computes the variance and CMA on the spectrums resulting from the successive interferograms detected around the same ZPD positions. Both amplitude and phase data are processed in parallel using dedicated logic: STAT Amp. and STAT Phase. As for the second stage, the statistics are independently computed for each spectrum position and temporarily stored in dedicated DDR memory segments between algorithm iterations.

The work in [210] describes an implementation of the CIRIS data processing stages on a Xilinx Zynq SoC. This SoC relies on using a DDR memory to exchange data between the different data processing stages. Namely, each processing stage is assigned a dedicated memory segment in the DDR and a data flow controller in the SoC coordinates all data transfers. However, this SoC does not use DPR, and hence cannot cope with any potential permanent damage provoked by cumulative radiation.

7.3 CIRIS Data Processing Tasks for Evaluation

The CIRIS data processing stages described in Section 7.2 are assigned to three different hardware tasks: ZPD, STAT and FFT. STAT is used three times, to process interferogram (STAT-I), amplitude (STAT-A), and phase data (STAT-P), respectively. The main functioning parameters for these hardware tasks are detailed in Table 7.1.

Table 7.1: Specification of the CIRIS tasks

Task	Execution time	Input Data	Output Data
STAT	100 µs	8,192 of 32bits (for 3 ports)	8,192 of 32bits (for 3 ports)
FFT	200 µs	8,192 of 18bits	8,192 of 32bits (for 2 ports)
ZPD	750 µs	13,000 of 18bits	8,192 of 18bits

The CELOC-based model requires each task to be interfaced with the network adapter of Figure 6.8 in Section 6.4, slightly increasing the resource utilization of each

task by 32 slices. Similarly, for the ICAP-based strategy, the task wrapper of Figure 3.5 presented in Section 3.4.4 is attached to each task to enable the tasks to use the configuration layer for data exchange as demonstrated in [79]. The increase in resource utilization here is only 21 slices and at least 2 BRAMs for the I/O data buffer. The static-communication implementation requires no wrapping and a bus-based network or a NoC is assumed. Though there is no wrapping, the network itself would consume a significant amount of resources with figures reaching 1669 FFs and 2035 LUTs for the switch block (with a 32-bit payload and 16-bit address) [184]. Table 7.2 shows the FPGA resource requirement of the hardware tasks, excluding the wrappers.

Table 7.2: Resources required by the CIRIS tasks

Task	**Flip-flops**	**LUTs**	**BRAM36s**	**DSP48s**
STAT	551	833	1	15
FFT	20,496	18,290	64	132
ZPD	1,055	9,694	12	32
Total	**22,102**	**28,817**	**77**	**179**

Figure 7.2 shows the CIRIS tasks implemented on the FPGA fabric. The biggest box in the bottom-left part of the FPGA is FFT and the smallest one in the bottom-right part is STAT. Note that for both CELOC and ICAP-based schemes tasks are self-contained within their boundaries, that is, they are free of external interconnections. The CAM occupies two different regions on the FPGA. The configuration manager and AXI peripherals are located next to the Arm cores in the top-left quadrant, and the DDR memory controller is located on the top-right part, close to the FPGA pin blocks to access the DDR memory chip. The top-right part also includes some static logic to interface with the CIRIS spectrometer and receive the interferogram samples, which are stored in the DDR memory.

The partitioning of the static logic in two locations is done very carefully to minimize the number of static routes. However, a significant number of routes is still needed to connect both static components in the two sides of the FPGA, as shown in Figure 7.2. As a result, the top 6 clock regions cannot be used to configure tasks. This

issue could be solved in a custom-made board by connecting the DDR memory chip to FPGA pins that are close to the Arm cores, in the top-left quadrant of the chip. This would free the entire right part for placing the CIRIS tasks.

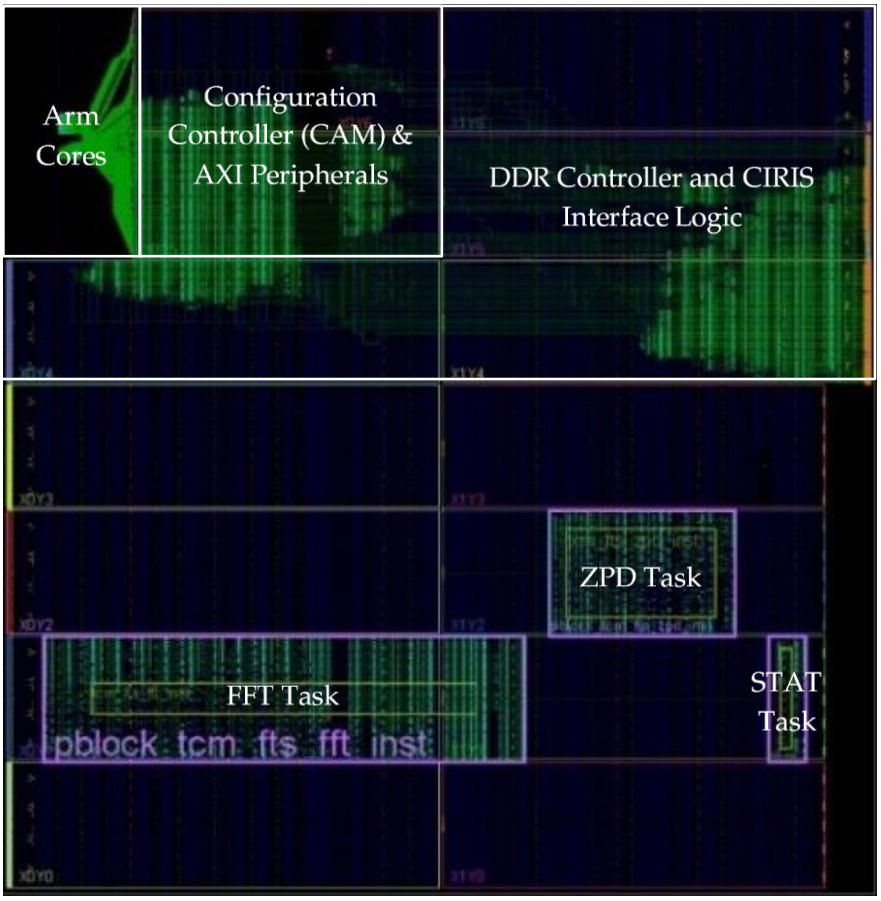


Figure 7.2: R3TOS-based CIRIS avionics system

7.4 CIRIS Avionics Models for Evaluation

In this study, the data flow controller in the SoC implementation described in [210] is assumed to be replaced by a ROS which manages task loading (provided by CAM) and inter-task communication. In order to demonstrate the capabilities of the contributions of this work, the configuration controller is used to configure the CIRIS tasks while inter-task communication is provided by a star-shaped CERANoC network. Reliability study is carried out to demonstrate that CELOC provides a better overall system reliability when compared to existing inter-task communication methods. Most of the existing RC systems use the static routing resources of FPGAs to implement P2P, bus,

and NoC structures (see Section 3.4), thereby hampering task relocation. On the other hand, the use of the configuration layer for communication highly favours relocation but at the expense of task configuration and SEM (see Section 3.4.4). It would be interesting to see how CELOC fairs with respect to these two approaches. Therefore, CELOC will be compared with a traditional fixed communication approach and an ICAP-based communication scheme using three CIRIS avionics models. A few simplifying, but nonetheless, practical and carefully-considered assumptions will be made along the line.

For all the models, the assumption is that none can keep multiple PBs of the same task. Moreover, a precise representation of the FPGA fabric is ensured by marking off regions occupied by static logic as well as hardware primitives such as the ICAP. All irregularities in the chip are also considered, including clock region boundaries and heterogeneous resource columns. The techniques described in [236][237] are used to choose the most optimal synthesis location for the tasks to improve their relocatability. That is, the synthesis positions for the tasks are selected not only to maximize the number of allowed relocation positions at runtime, but also to minimize overlapping among them, as described in [237].

7.4.1 Static CIRIS Avionics Model

In the static CIRIS avionics model, the available area in the chip is distributed into 16 different slots as shown in Figure 7.3. Tasks are mapped to slots in one-to-one fashion. The smallest A-type slots can host only STAT tasks, medium size B-type slots can allocate both STAT and ZPD tasks, and the largest C type slots can allocate any of the tasks. The most optimistic case in which no static routes go beyond the static region is considered. It should be noted that this is not the case in most slotted systems that use static communication infrastructures. Besides, the CIRIS controller in [210] does not use slots at all.

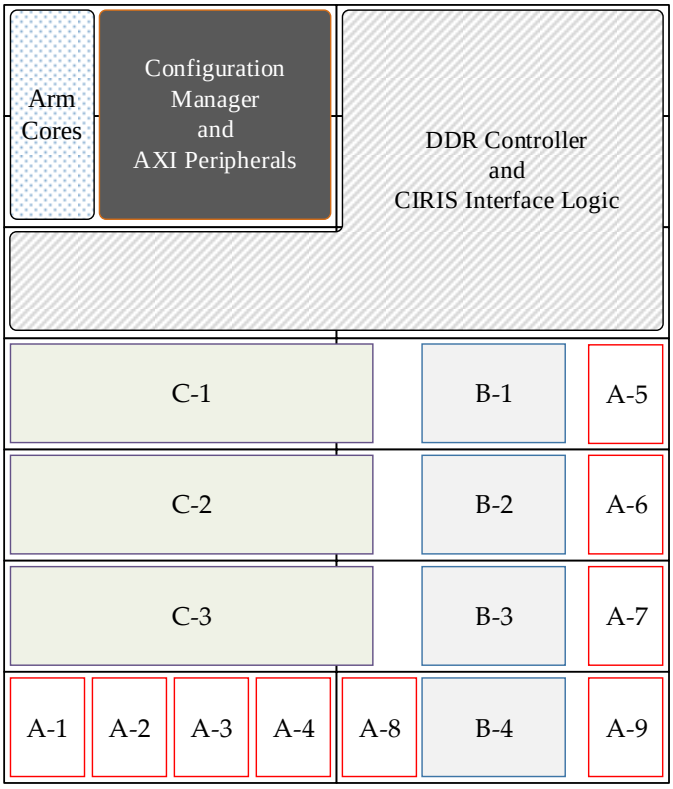


Figure 7.3: Static CIRIS avionics model

7.4.2 ICAP-Based CIRIS Avionics Model

Figure 7.4 shows the initial task allocation in the ICAP-based CIRIS avionics model. This is also the synthesis location for the tasks. Note that this initial task allocation is exactly the same as in the static model to allow a fair comparison between them. At runtime, tasks can be relocated to any position on the FPGA where the arrangement of the resources matches that in the original synthesis location.

7.4.3 CELOC CIRIS Avionics Model

For CELOC, the reconfigurable region is partitioned into circuit regions (CRs) and used as nodes. Unlike in the 4-node star network of Section 6.6 that uses a whole clock region as node, the CERANoC network here hosts two nodes in a single clock region resulting in 15 nodes and one *Central Router* as shown in Figure 7.5. The versatility of CELOC brought about by the variety of clock buffer combinations can be exploited in this model. CRs to the outer edges of the device use the buffer configuration [BUFR→

BUFG→] for accessing the CERANoC network while those to the inner edges use [BUFG→]. It should be noted that the meeting point between the two CRs in a clock region is not a fixed one. The tasks are flushed to left and right to access the buffers. As such, it is possible for both A- and B-type location pairs to coexist in the clock regions on the right of the chip.

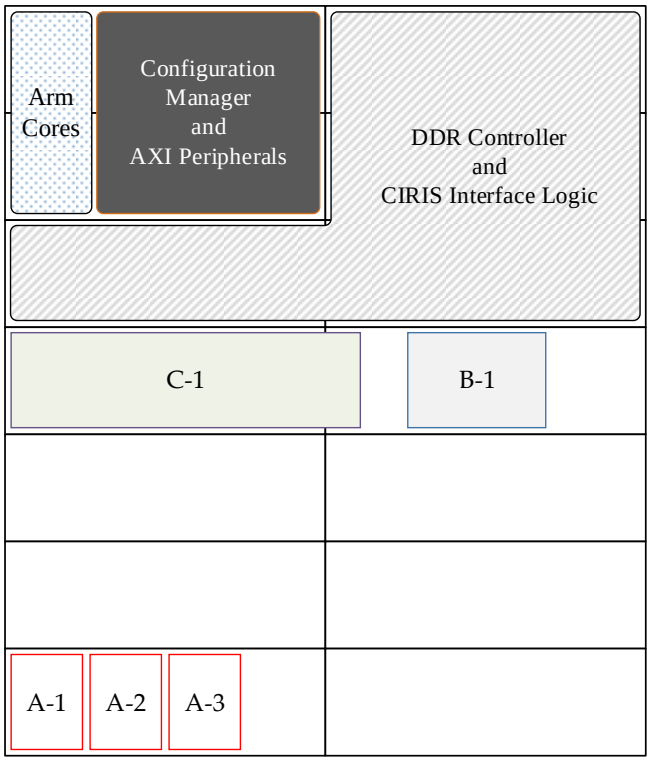


Figure 7.4: ICAP-based CIRIS avionics model (initial configuration)

Meanwhile, it should be noted that the entire right side of the reconfigurable region in the target board has no BUFRs and BUFMRs. To allow for a fair comparison, the buffers are assumed to be present as they would be in other chips and meanwhile any application deploying CELOC would specifically choose a chip that has all or most of its clock buffers in place. Nevertheless, if this particular chip were to be used in reality, the left and right nodes in the right clock regions can use a horizontal partition rather than a vertical one, with each directly accessing a BUFG. However, tasks occupying those nodes would have to take flat rectangular shapes and care must be taken to properly preserve the other node when one node is been reconfigured. It should recalled that configuration frames are aligned vertically to clock region heights. While this task

partition could have been used in Figure 7.5 that has not been done because the tasks are laid out to ensure the lowest fragmentation (the presence of unused resources) possible and an optimum relocatability for each task. Note the same starting positions are used for the tasks as with the Static and ICAP-based avionics.

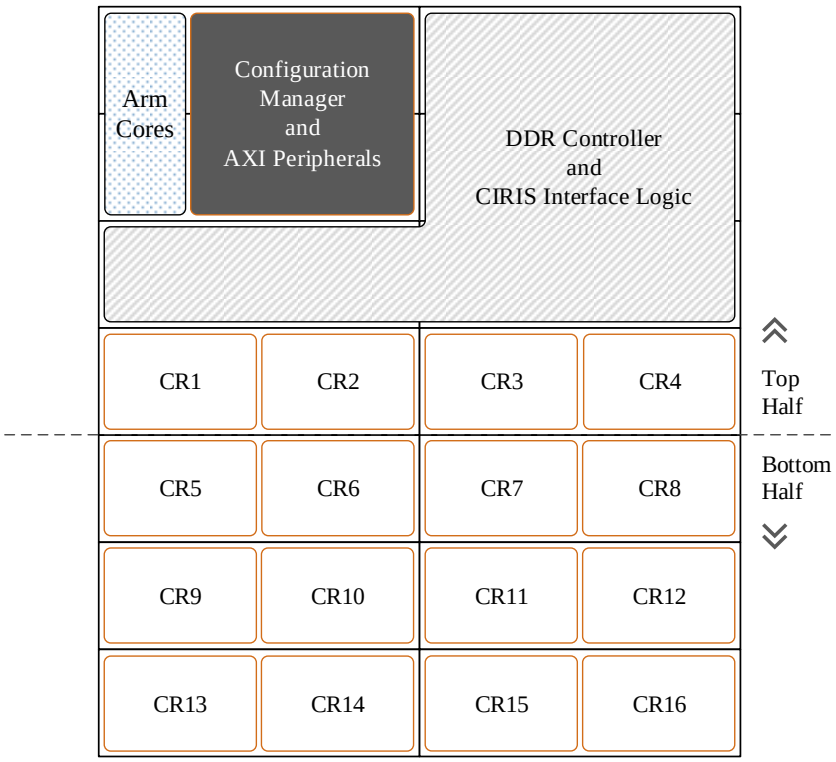


Figure 7.5: CELOC-based CIRIS avionics model

7.5 Inter-Task Communication Evaluation

Figure 7.6 depicts the data flow between the tasks, which execute concurrently in pipeline on the FPGA. Table 7.3 shows the data communication latencies for the three avionics models. For the static communication, the latencies are estimated from the AXI bus-based latencies in [210]. The results are indicated in the "Static Links" column in Table 7.3.

In the ICAP avionics model, a single BRAM36 is used for each of the IDM and ODM (see Figure 3.5 in Section 3.4.4). As a result, the ICAP-based data transfer throughput is 7.805 MB/s based on the use of the RMW operation (see Section 4.2.4) for data relocation. This latency can be improved to as high as 78.05 MB/s for 10

BRAM36s but this would have a serious impact on the area utilization. In fact, each CIRIS task would need an additional 18 BRAM36s, 9 on the input and 9 on the output. This translates to 2 additional BRAM columns for each of the tasks. These columns cannot be absorbed into the current RP area coverages. In addition, more non-BRAM resources would be wasted as it is in general, impossible to select BRAM-only columns and certainly, there are no BRAM-BRAM column pairs.

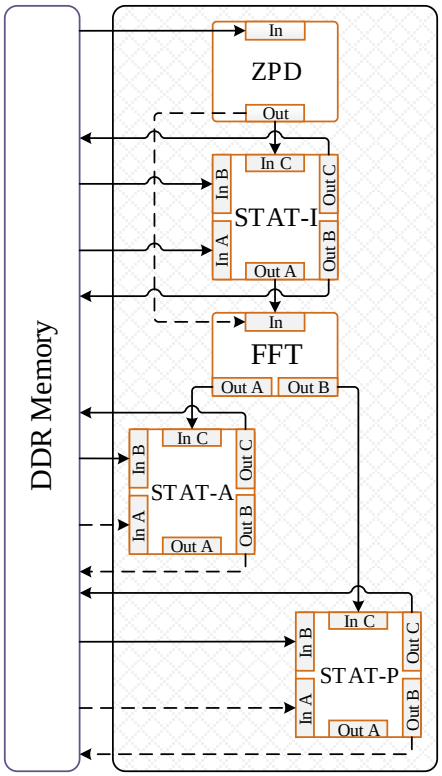


Figure 7.6: Simplified data flow of the CIRIS avionics

From Section 6.5.5, the average network speed of CERANoC for the CELOC-based model is 379.134 MHz, calculated from 70.21% of f_{BUFR_MAX}, where f_{BUFR_MAX} is 540 MHz for the Kintex-7 at speed grade -2 (see Table 6.7).. Therefore, for the 15-node CERANoC clocked at 379.134 MHz, the total network throughput is determined from Equation (6.1) as 2,843.505 Mbps (355.44 MB/s) for a 32-bit payload per packet. Each node contributes an average of 23.70 MB/s (24,264.58 kB/s) to this total. Note that the address field in the packet is still 4 bits. A 4-bit field can address 16 nodes. Since a zero address is avoided (refer to Section 6.6.1) and a node in the 16 circuit regions is used as the *Central Router*, the remaining 15 nodes can be addressed from 0x1 to 0xF.

Table 7.3: Inter-task latencies for the different communication mechanisms

Communication Flow	Data Size (Bytes)	Latency (ms)		
		CELOC	*ICAP-Based*	*Static Links*
DDR Memory → ZPD	29,250	1.180	3.574	0.100
ZPD → STAT-I	18,432	0.744	2.252	0.041
DDR Memory ↔ STAT-I	65,536	2.644	8.008	0.188
STAT-I → FFT	18,432	0.744	2.252	0.100
FFT → STAT-A	32,768	1.322	4.004	0.041
FFT → STAT-P	32,768	1.322	4.004	0.041
DDR Memory ↔ STAT-A	65,536	2.644	8.008	0.188
DDR Memory ↔ STAT-P	65,536	2.644	8.008	0.188
Total	**328,258**	**13.244**	**40.11**	**0.887**

Since this is a data processing application where a new interferogram is received every 24.8 ms, it is extremely important to meet real-time performance. Note that CERANoC requires 13.244 ms to move data for a complete processing of an interferogram. This is 3.03x better than an ICAP-based inter-task communication which takes up to 40.11 ms, which is already higher than the interval of interferogram arrival. Because of the bit-parallel nature of data transfer in the static approach, it requires only 0.887 ms for communication.

7.6 Reliability Study

This study will evaluate the implication of the different communication approaches on the reliability of the CIRIS avionics system. Two studies will be carried out, one for soft error mitigation evaluation and the other for permanent damage (hard error) mitigation evaluation. The performance of each avionics model will be evaluated.

7.6.1 Soft Error Mitigation Evaluation

First, the SEM scanning time for each of the task has to be determined using Table 4.15 from Section 4.5.2. The entire area occupied by each task has to be scanned in each SEM cycle. However, since it is impossible to partition the CIRIS tasks to occupy areas

spanning only the resources in Table 7.2, it becomes necessary to align the RPs to reconfiguration frames spanning columns of CLB-CLB, CLB-BRAM, and CLB-DSP pairs (see Section 2.2.2). The resulting area occupation in terms of columns of CLBs, BRAMs, and DSPs is presented in Table 7.4 along with the number of frames needed for SEM. Note that the BRAMs are not scanned; therefore, the number of SEM frames is less than the number of frames occupied by the task. The configuration (CFG) and SEM scan times are also included, with a complete SEM scan cycle for all the tasks requiring a little over 4 ms.

Table 7.4: Area overhead of the CIRIS tasks

Task	**Number of Columns**			**#Occupied Frames**	**#SEM Frames**	**CFG Time (µs)**	**SEM Scan Time (µs)**
	CLB	***BRAM***	***DSP***				
STAT-I	4	1	1	300	172	306.72	175.2
STAT-A	4	1	1	300	172	306.72	175.2
STAT-P	4	1	1	300	172	306.72	175.2
FFT	62	7	9	3,380	2,484	3,417.52	2,510.32
ZPD	25	2	3	1,240	984	1,256.12	995.32
Total	**99**	**12**	**15**	**5,520**	**3,984**	**5,593.8**	**4,031.24**

Table 7.5 summarizes the total required configuration memory access times for all the strategies. The configuration times and SEM scan times are the same for all the three strategies. The CELOC and ICAP-based strategies incur resource overheads for their network adapters. However, these are easily absorbed into the slice usage of the tasks themselves and as such, do not increase configuration and SEM overhead. If the ICAP's available time is normalized to the total time (26.81 ms) required by the CIRIS application, then it implies that the CELOC-based avionics model has an occupation of the ICAP at 64.12% (2.79x) less for the CIRIS application when compared to the ICAP-based model.

The simplifying assumption is made that there is frequent need for reconfiguring (relocating) the tasks to improve reliability by wear levelling (see Section 3.2.3 for a discussion on wear levelling). For the ICAP-based model, all the operations can only

be done in sequence using the single available ICAP. Therefore, if we assume that the system operations always follow the cycle of task (re)configuration, SEM, and inter-task communication, then it becomes clear that one of these critical functions would suffer. The full SEM cycle scan (based on the selective-area scanning introduced in Section 4.3.2) is 4.03 ms. If this is taken as 99% as recommended by Xilinx to ensure reliability [187], then only 40.71 µs is available for the other functions, that is, configuration in the case of CELOC and Static Bus/NoC CIRIS avionics; and both configuration and communication for the ICAP-based model. Communication alone in the ICAP-based CIRIS avionics requires 40.11 ms, which is 985x of 40.71 µs. Assuming inter-task communication cannot be delayed for the reason of another reliability-related factor, that is, real-time deadlines, then either SEM or reconfiguration would suffer.

Table 7.5: ICAP bandwidth utilization of the CIRIS tasks

System Model	ICAP Time Required (ms)			Total	% of Total
	CFG	SEM	Comm.		
CELOC	5.59	4.03	0	9.62	19.33%
ICAP-Based			40.11	49.78 (x5.17)	80.65%
Static Bus/NoC			0	9.62	19.33%
Total	49.78 ms				

7.6.2 Hard Error Mitigation Evaluation

For the purpose of comparison, the resilience to permanent damages of the three avionics models can be evaluated by estimating how many such damages each model can tolerate before failing. A good estimate can be obtained for each avionics model by counting how many locations each of the CIRIS tasks can be relocated to while maintaining full communication access as permanent damages emerge. For a fair comparison, it is assumed that none of the avionics models can keep multiple PBs of the same task. If a region occupied by a task fails, online relocation is used to configure it somewhere else. Table 7.6 indicates the performance metrics of the avionics models.

The number of overlapping locations (#LOC) for each task is reported. The bigger a task is, the more prone it is to being hit by an error requiring it to be relocated. This is because the task occupies a larger area. This is taken into account by normalizing the largest task's size to 1, and obtaining size factors that are multiplied with #LOC to get a fairer result for the relocation (RELOC) quality which reflects the reliability of the model.

Referring to the Static avionics model in Figure 7.3, the provision for relocation involves activating another instance of a failed task only in the location determined at compile time. There is no possibility for space-multiplexing the tasks to share chip area. Therefore, once a task fails, another instance can be used in its place until all the instances for any one task has failed, at which point the system fails. The FFT only has 3 locations to use while the ZPD has 4. All the a-type slots are shared 3 each by the STAT tasks.

On the other hand, both the CELOC and ICAP-based avionics model can withstand more damages since tasks can be relocated to any position on the FPGA where there is a matching arrangement of the resources. The sheer size of the FFT task means that it can only use one extra location in addition to the initial 3. It can use the location occupied by A-1 to A-8, whereas the ZPD can use its own 4 locations and an additional 8 from the left clock regions in the reconfigurable region.

Table 7.6: Reliability performance of the avionics models

Task	**Size Factor**	**Static Avionics**		**ICAP Avionics**		**CELOC Avionics**	
		#LOCs	***RELOC***	***#LOCs***	***RELOC***	***#LOCs***	***RELOC***
STAT-I	0.089	3	0.267	30	2.67	30	2.67
STAT-A	0.089	3	0.267	30	2.67	30	2.67
STAT-P	0.089	3	0.267	30	2.67	30	2.67
FFT	1.000	3	3.000	4	4.00	4	4.00
ZPD	0.367	4	1.468	12	4.40	12	4.40
Total		**16**	**5.269**	**106**	**16.41**	**106**	**16.41**
Comparison		**x1**	**x1**	**x6.63**	**x3.11**	**x6.63**	**x3.11**

For the CELOC and the ICAP-based CIRIS avionics, the ZPD task can use up to 12 locations. However, this number could have been 8 for CELOC considering that by being closer to the left inner edge of the clock regions, the ZPD tasks would have been routed to access BUFGs for communication. However, note that the CELOC network adapter can be used such that the CE connection to the clock buffers is effected in runtime by simply activating the required PIPs. This does not require an expensive online clock routing as the buffer configuration needed for each CR can be laid offline by using dummy loads (see Section 3.2.5) [154].

Although there is an unavoidable mutually-exclusive relationship between the A-, B-, and C-type slots in every row for CELOC. For instance, if the C-2 slot is occupied by the FFT task, then an A-type slot would have to be sacrificed in order to access BUFMRs and BUFRs on the right of the clock region. That is, all the three cannot be placed in a single row at the same time. Such a scenario does not occur with the ICAP-based avionics. However, this is negligible compared to the immense overall system reliability enabled by CELOC, especially in terms of soft error mitigation, which is the prevalent failure mode in reconfigurable hardware systems.

Overall, for the CIRIS data processing, CELOC proves to be better for a holistic approach to reliability. Indeed, in space-grade systems like the CIRIS, system reliability is crucial. While process-based radiation hardening can be used to improve reliability in a static CIRIS avionics implementation, and a configuration-layer-based communication used to enable a 3.11x tolerance to permanent damages, CELOC is a cheaper and a more reliable option as it enables tolerance to both soft and hard errors in COTS reconfigurable device. In fact, compared to the non-slotted static implementation of the CIRIS controller in [210], CELOC offers a reliability improvement of 16.41x. While a much rigorous testing like fault injection analysis would give a clearer picture, CELOC can still be expected to outperform both static and ICAP-based communication schemes in terms of reliability.

7.7 Chapter Summary

In this chapter a case study application has been presented to demonstrate the advantage of adapting clock buffers and nets for on-chip communication. The evaluation drew a case study from a NASA/JPL spectrometer data processing controller. The stages of this data processing controller were used as tasks and configuration, communication, and soft error mitigation times were determined. The application highlighted that CELOC is 3.03x faster when compared to a configuration–layer-based communication. However, as expected, it is slower when compared to a static implementation. What sets CELOC apart from the static communication architecture is that it supports relocation inherently. While the ICAP-based model is able to relocate the tasks more freely compared to CELOC, from a holistic system perspective, CELOC provides an improved overall system reliability.

Chapter 8

Conclusions and Future Work

8

The core of this thesis proposed and implemented novel methods and strategies that could serve as the frameworks for reliable, high-performance, available, efficient, and secure real-time reconfigurable computing. The observed trend in computing is the change in paradigm from processor-only computation to hardware-assisted accelerated computing, where software processes running on a CPU are accelerated by offloading compute-intensive and time-critical tasks to hardware. The prominent deployed hardware solution is the FPGA, where hardware tasks (circuits) are dynamically managed by a *Reconfigurable Operating System* (ROS) on behalf of a host software operating system. The key system services or frameworks in a ROS are task configuration and inter-task communication. However, in order to ensure system reliability, especially for the high-end applications like aerospace, military, defence, and nuclear that now use the FPGA, these services are required to be real-time, efficient, and secure. These two key services have been the subject of this thesis and the core contributions of this work have been presented in Chapters 4 through 6 while Chapter 7 evaluated the proposed frameworks with a single application by drawing a case study from the NASA/JPL's CIRIS instrument.

8.1 Summary, Limitations, and Concluding Remarks

Chapter 4 presented a high-performance and reliability-centric configuration memory access controller with key features for task management in reconfigurable systems. In the light of the aims and objectives specified in Section 1.2, the proposed configuration controller has proven to be both resource- and time-efficient. It saves up to 71% area and has 30% less configuration latency in comparison to state-of-the-art implementations. In addition, with an average raw throughput of 380 MB/s, which is very close to the theoretical maximum of 400 MB/s, the controller has demonstrated a very high performance. Moreover, the fact that configuration errors are monitored and

provisions made for runtime correction, where possible, implies that the controller is able to improve the availability of the device for high-end applications like datacentres and aerospace. Related to this, the provision of soft error detection and correction functionalities will be invaluable in devices that operate in the harsh environment of space and nuclear decommissioning. Indeed, based on the observation with the NASA/JPL CIRIS data processing case study, the controller can mitigate errors at up to 74% less time compared to the state-of-the-art vendor SEM IP.

The runtime relocation of circuits is a key mechanism for improving the reliability of reconfigurable devices. However, the vendor bitstream format for encrypted bitstreams hampers the relocation of circuits by including the frame address as cyphertext in the bitstream. The frame address specifies the location where a circuit should be configured and for relocation, this needs to be manipulated in plain format. In Chapter 5, a completely new format for encrypted bitstreams was proposed. This relied on the fact that a plain frame address can be loaded on-chip in advance of the frame data that defines the functionality of the circuit being configured. The concept was thus termed *Advance Task Address Loading* (ATAL). An algorithm was developed (with a Windows-based GUI designed) for parsing a partial bitstream generated by Vivado and formatting it to remove the frame addresses and splitting the bitstream into multiple parts which can be loaded separately after the on-chip loading of runtime-generated plain frames addresses. ATAL has shown promising results by outperforming other strategies for encrypted partial bitstream relocation, providing a far more time-efficient relocation with very small area overhead. In fact, the only area overhead was that of the configuration controller, which was needed whether or not ATAL was used. As such, ATAL can be considered to have no resource overhead. In order to load the specially-formatted ATAL bitstream, a unique configuration controller was developed.

The most important limitation of the controllers developed in Chapter 4 and Chapter 5 is the configuration speed, which is due to the limited speed of the configuration interface of the device used in the prototype. While the logic resources and the FPGA in general can operate at clock frequencies that are a few hundreds of MHz, the configuration interface is limited to 100 MHz. Similarly, the ICAP also posed

a limitation in the ATAL configuration controller. Apart from the limited speed of 100 MHz, the ICAP only supports bitstream loading though its 8-bit bus when encryption is used. This means that the configuration latency for encrypted bitstreams is in general, quadrupled. Moreover, when an encrypted bitstream is being loaded it is not straightforward to know when configuration has been concluded, making the design of the controller more complicated by having to keep track of how many words has been loaded.

Certainly, the configuration interface in FPGAs and the provision of security should not undermine the high performance that has come to be expected of hardware. As a result, some of these issues have been addressed to some extent in the latest device families. The ICAP in the UltraScale architecture can be clocked at up to 200 MHz and in fact, it only supports the 32-bit interface, even with encryption used, and has extra signal ports for configuration monitoring. It would be interesting therefore, to port these controllers to the new architecture for an improvement on an already impressive performance.

While the controllers have been targeted at the Xilinx 7 series architecture, only minor modifications are needed in order to adapt them for use in the UltraScale architecture. For devices from Altera and other FPGA manufacturers, the same methodologies can be followed – build a custom control structure around the configuration interface to deliver reliability-enabling functionalities. For ATAL's configuration controller, an insight would have to be gained into the encryption and authentication scheme adopted by the manufacturer. However, the most important aspect is to be able to leave the frame address out of the encrypted portion of the configuration bitstream.

In Chapter 6, a new mechanism for on-chip communication was advanced. Motivated by the need to provide a dynamic communication support for relocatable circuits, the concept involved the adaptation of clock buffers and nets in a typical FPGA for linking nodes in a NoC. This approach to communication was characterized and used to demonstrate relocation. However, because of the serial nature of the data transmission, the overall throughput of a network so formed is limited by the maximum usable operating frequency that does not violate setup and hold times. In addition, the

fact that a transmitter needs access to the CE pin has restricted relocation of tasks to clock-region-sized partitions in the prototype implementation. If a task is not big enough, there could be many unused resources in the region. One way to resolve this is to use a multiplexing approach to the connection to the CE pins of the buffers in order to allow multiple tasks to share the same CE. This will ensure that a small task placed in a clock region does not prevent other tasks from using the same region. This would reduce the link bandwidth available to each task. On the other hand, a better approach would be to have multiple horizontal tasks in the same region accessing BUFHs to form a mesh network.

Moreover, one limitation in the present implementation of CELOC/CERANoC arises from the limited number of clock buffers, limiting the network throughput. This could possibly be alleviated in newer chips like the UltraScale, which have more clock buffers. This is worth investigating in the future.

8.2 Recommendations for Future Work

Having designed and implemented the supporting infrastructures for reliable real-time reconfigurable computing, the next stage, which could serve as future work would be the development of a truly reliable RC system. This would aim at bringing all the mechanisms advanced in this thesis together into a complete system architecture for RC, with a more holistic approach given to reliability, real-time processing, high performance, availability, efficiency, and security. Shown in Figure 8.1 are key system components that will be needed for effecting the suggested system for future work.

The system is proposed to be made up of two major sub-systems: a *Software Microkernel* (SWμK) and a *Hardware Microkernel* (HWμK). The SWμK would contain the Main CPU for executing software tasks and a *Scheduler* and an *Allocator*, which can also be implemented in hardware. The HWμK would use the reconfigurable fabric to implement system managers and provide a chip area for placing hardware tasks.

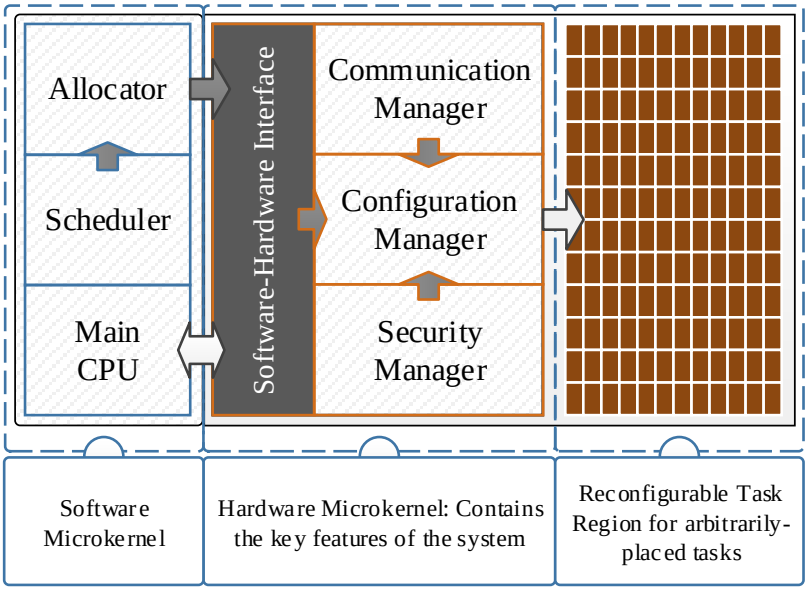


Figure 8.1: System architecture for reliable reconfigurable computing

The key components of the recommended system and other relevant features are discussed below:

- The core of this system would be a *Configuration Manager* based on the controller described and implemented in Chapter 4, providing salient system functionalities that include task loading, task deallocation, task replication for TMR implementation, device error diagnosis, and SEM.
- Inter-task communication and synchronization would be provided in the clock routing layer through the instrumentality of the clock buffers and nets in an FPGA. The methods advanced in Chapter 6 would be highly relevant for this, especially for providing dynamic communication support for PBR.
- A *Communication Manager* would provide communication support at the system level, bridging synchronizations between the hardware tasks and parent software processes executing on a Main CPU.
- A *Scheduler*, which determines the order of task execution would also be needed. Depending on the targeted application, the scheduling may be based on *Earliest Deadline First* (EDF) [188], which is a popular scheduling approach for real-time computing.

- To find a suitable region for a hardware task on the chip, an *Allocator* would be necessary. With considerations given to the reduction of chip area fragmentation, methods like those proposed in [237], [238] and [239] can be used. A *Security Manager* would cater for both task- and system-level security using the approaches developed in Chapter 5.

8.2.1 Traditional Slotted Reconfigurable Systems

In a traditional partially reconfigurable system, each reconfigurable circuit is synthesized using resources exclusively contained within a predefined rectangular region in the FPGA. That FPGA region is named as "reconfigurable slot". Several reconfigurable circuits can be mapped to the same slot at design time. At runtime, it is possible to switch between the different partial bitstreams associated to the reconfigurable circuits to change the functionality implemented in the FPGA slots [37]. Therefore, slotted reconfigurable systems exhibit coarse granularity defined by the slots in which the FPGA is divided.

Two aspects need to be considered here. First, in addition to the configuration information for the FPGA resources contained in the slots, partial bitstreams also include configuration information associated with static routes that cross the slots connecting resources located in non-reconfigurable FPGA regions. FPGA vendor tools ensure that these static routes are always preserved in all generated partial bitstreams. Second, Partition Pins (PartPins) are used to preserve the interfaces of the slots in all possible configurations [37]. PartPins are LUTs inserted by the build tool to connect resources located in both reconfigurable and static FPGA regions. Therefore, PartPins shape the physical boundaries of slots, which are decided at design time and remain fixed at runtime.

8.2.2 Towards Slotless Reconfigurable Systems

Partial bitstreams can be relocated to different FPGA regions where resources are arranged in the same exact way as in the reconfigurable slots for which the partial bitstreams were originally generated (refer to Section 3.2.4). Note that this can be done

by changing the configuration frame FAR addresses in the partial bitstreams to refer to the target relocation regions.

However, this process is not currently supported by any FPGA vendor tool and therefore, there are some limitations as pointed out in Section 3.2.4. The root problem of this is the use of static routing resources for inter-task communication (see Section 3.1). While a recent ROS, in the name of R3TOS manages to avoid the use of static interconnections, by using the configuration infrastructure of the FPGA for inter-task data transfer, it risks upsetting the same reliability for which it is developed. Other critical services (e.g., SEM and task loading) relying on configuration are impaired, as the configuration interface is a single resource that needs to be shared by these critical system services.

8.2.3 Partition Architecture for Reliable Computing

To avoid static routes, a partition architecture shown in Figure 8.2 can be used for placing tasks in the proposed system. In this architecture, intra-clock region communication is enabled by BUFHs while inter-clock region communication is achieved through a combination of BUFMRs and BUFRs. For generalization, it should be noted that the indicated clock buffers can be easily replaced with equivalent primitives in other FPGAs and PLDs, whether from Xilinx or from other manufacturers.

To avoid static routes, RePARC does not rely on the general routing for inter-communication and synchronization, save for a single wire used for accessing the CE of a clock buffer. Hence, it keeps the FPGA’s reconfigurable area generally empty, that is, free of any partition boundaries (i.e., PartPins) and static routes. Inter-task communication and synchronization are carried out through the FPGA's clock routing layer, as described in Chapter 6. In this context, task allocation and deallocation are considerably accelerated, as there is no need to preserve any static routes in the target and source positions when relocation is being performed. In fact, task deallocation simply consists in blanking the whole content of the corresponding frames, which can be done very quickly using bitstream compression DPR commands.

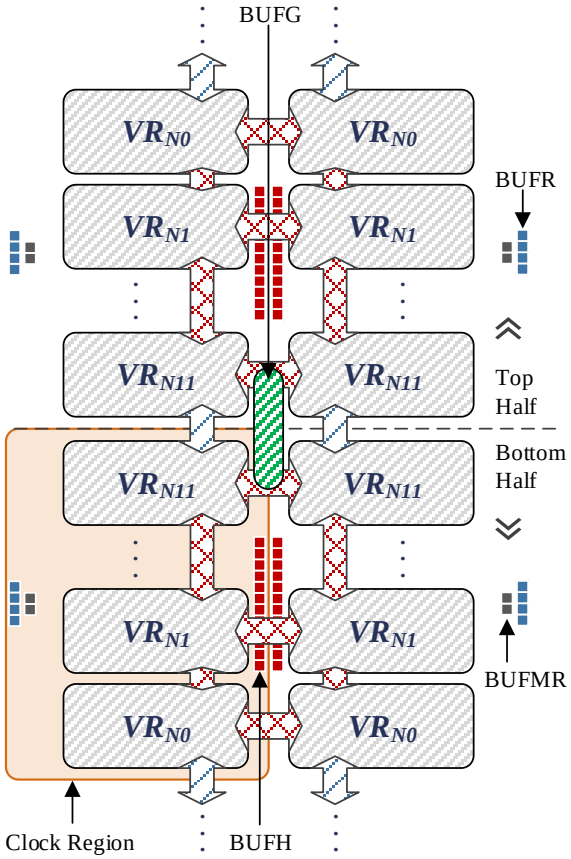


Figure 8.2: Partition architecture showing slots interconnected by clock buffers (represented by the arrows)

References

[1] C. Kachris and D. Soudris, ‘A survey on reconfigurable accelerators for cloud computing’, in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–10.

[2] P. Alfke, I. Bolsens, B. Carter, M. Santarini, and S. Trimberger, ‘It’s an FPGA!’, *IEEE Solid-State Circuits Mag.*, vol. 3, no. 4, pp. 15–20, 2011.

[3] A. Putnam *et al.*, ‘A reconfigurable fabric for accelerating large-scale datacenter services’, in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 13–24.

[4] Amazon Inc., ‘Amazon EC2 F1 Instances’, *Amazon Web Services, Inc.* [Online]. Available: //aws.amazon.com/ec2/instance-types/f1/. [Accessed: 01-Aug-2017].

[5] J. Backus, ‘Can Programming Be Liberated from the Von Neumann Style?: A Functional Style and Its Algebra of Programs’, *Commun ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978.

[6] G. E. Moore, ‘Cramming More Components Onto Integrated Circuits’, *Proc. IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.

[7] G. E. Moore, ‘Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.’, *IEEE Solid-State Circuits Soc. Newsl.*, vol. 11, no. 3, pp. 33–35, Sep. 2006.

[8] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 2008.

[9] D. Koch, D. Ziener, and F. Hannig, ‘FPGA Versus Software Programming: Why, When, and How?’, in *FPGAs for Software Programmers*, D. Koch, F. Hannig, and D. Ziener, Eds. Cham: Springer International Publishing, 2016, pp. 1–21.

[10] S. M. Trimberger, ‘Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology’, *Proc. IEEE*, vol. 103, no. 3, pp. 318–331, Mar. 2015.

[11] S. Welsh and P. Knaggs, ‘ARM Assembly Language Programming’. 2003.

[12] G. M. Hopper and R. R. Corp, ‘The Education of a Computer’, in *Proceedings of the 1952 ACM National Meeting*, Pittsburgh, Pennsylvania, 1952, pp. 243–249.

[13] R. K. Ridgway, ‘Compiling Routines’, in *Proceedings of the 1952 ACM National Meeting*, Toronto, Ontario, Canada, 1952, pp. 1–5.

[14] G. Martin and G. Smith, ‘High-Level Synthesis: Past, Present, and Future’, *IEEE Des. Test Comput.*, vol. 26, no. 4, pp. 18–25, Jul. 2009.

[15] Xilinx Inc., ‘Vivado Design Suite User Guide, High-Level Synthesis - UG902 (v2017.4)’. Xilinx Inc., 2018.

[16] R. Nane *et al.*, ‘A Survey and Evaluation of FPGA High-Level Synthesis Tools’, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[17] R. Gupta and F. Brewer, ‘High-Level Synthesis: A Retrospective’, in *High-Level Synthesis*, Springer, Dordrecht, 2008, pp. 13–28.

[18] D. G. Bailey, ‘The advantages and limitations of high level synthesis for FPGA based image processing’, in *Proceedings of the 9th International Conference on Distributed Smart Camera - ICDSC ’15*, Seville, Spain, 2015, pp. 134–139.

[19] G. Brebner, ‘A virtual hardware operating system for the Xilinx XC6200’, in *Field-Programmable Logic Smart Applications, New Paradigms and Compilers*, vol. 1142, R. W. Hartenstein and M. Glesner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 327–336.

[20] G. Wigley and D. Kearney, ‘The first real operating system for reconfigurable computers’, in *Proceedings of the 6th Australasian Computer Systems Architecture Conference. ACSAC 2001*, Gold Coast, Qld., Australia, 2001, pp. 130–137.

[21] J. Tørresen and D. Koch, ‘Can Run-time Reconfigurable Hardware be more Accessible?’, in *International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2011.

[22] G. Wigley and D. Kearney, ‘The Development of an Operating System for Reconfigurable Computing’, in *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’01)*, 2001, pp. 249–250.

[23] G. Wigley and D. Kearney, ‘Research issues in operating systems for reconfigurable computing’, in *proceedings of the International Conference on Engineering of Reconfigurable System and Algorithms (ERSA)*, 2002, pp. 10–16.

[24] M. Eckert, D. Meyer, J. Haase, and B. Klauer, ‘Operating System Concepts for Reconfigurable Computing: Review and Survey’, *Int. J. Reconfigurable Comput.*, pp. 1–11, 2016.

[25] R. Katz, ‘The failure of a small satellite and the loss of a space science mission’, in *Proceedings 2002 NASA/DoD Conference on Evolvable Hardware*, 2002, p. 4.

[26] T. Bayer, B. Cooke, I. Gontijo, and K. Kirby, ‘Europa Clipper mission: the habitability of an icy moon’, in *2015 IEEE Aerospace Conference*, 2015, pp. 1–12.

[27] R. B. Gardenyes, ‘Trends and patterns in ASIC and FPGA use in space missions and impact in technology roadmaps of the European Space Agency’, Delft University of Technology, Delft, The Netherlands, 2012.

[28] J. H. Yuen, Ed., *Deep Space Communications*. New Jersey: John Wiley & Sons, Inc., 2016.

[29] FAA, ‘Established Practices for Human Space Flight Occupant Safety’. 2014.

[30] S. B. Johnson, ‘Reliable avionics design for deep space’, in *9th IEEE/AIAA/NASA Conference on Digital Avionics Systems*, 1990, pp. 35–40.

[31] D. Sinclair and J. Dyer, ‘Radiation effects and COTS parts in SmallSats’, in *Proceedings of the 27th Annual AIAA/USU Conference on Small Sattellites*, 2013.

[32] M. E. Pate-Cornell, R. L. Dillon, and S. D. Guikema, ‘On the Limitations of Redundancies in the Improvement of System Reliability’, *Risk Anal.*, vol. 24, no. 6, pp. 1423–1436, Dec. 2004.

[33] R. P. Ocampo, ‘Limitations of spacecraft redundancy: A case study analysis’, 2014.

[34] M. Armbrust *et al.*, ‘A View of Cloud Computing’, *Commun ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
[35] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, and C. Mcdermid, ‘Availability and load balancing in cloud computing’, in *International Conference on Computer and Software Modeling, Singapore*, 2011, vol. 14.
[36] B. Hayes, ‘Cloud computing’, *Commun. ACM*, vol. 51, no. 7, pp. 9–11, 2008.
[37] Xilinx Inc., ‘Vivado Design Suite User Guide, Partial Reconfiguration - UG909 (v2018.1)’. Xilinx Inc., 2018.
[38] J. A. Stankovic and K. Ramamritham, ‘What is predictability for real-time systems?’, *Real-Time Syst.*, vol. 2, no. 4, pp. 247–254, Nov. 1990.
[39] A. Putnam *et al.*, ‘A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services’, *IEEE Micro*, vol. 35, no. 3, pp. 10–22, May 2015.
[40] J. Ouyang, ‘SDA: Software-defined accelerator for large-scale deep learning system’, in *2016 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2016, pp. 1–1.
[41] J. Ouyang, S. Lin, W. Qi, Y. Wang, B. Yu, and S. Jiang, ‘SDA: Software-defined accelerator for large-scale DNN systems’, in *2014 IEEE Hot Chips 26 Symposium (HCS)*, 2014, pp. 1–23.
[42] H. Chauhan, ‘Can Intel Dominate This Market by Overcoming This Smaller Rival? -- The Motley Fool’, Nov-2017. [Online]. Available: https://www.fool.com/investing/2017/11/24/can-intel-dominate-this-market-by-overcoming-this.aspx. [Accessed: 16-Aug-2018].
[43] S. M. Trimberger and J. J. Moore, ‘FPGA Security: Motivations, Features, and Applications’, *Proc. IEEE*, vol. 102, no. 8, pp. 1248–1265, Aug. 2014.
[44] Xilinx Inc., ‘7 Series FPGAs Clocking Resources - User Guide UG472 (v1.11.2)’. Xilinx Inc., 2015.
[45] J. Rose and S. Brown, ‘Flexibility of interconnection structures for field-programmable gate arrays’, *IEEE J. Solid-State Circuits*, vol. 26, no. 3, pp. 277–282, Mar. 1991.
[46] N. Mehta, ‘Xilinx 7 Series FPGAs: The Logical Advantage’. 2012.
[47] Xilinx Inc., ‘7 Series FPGAs Configuration, User Guide - UG470 (v1.13.1)’. Xilinx Inc., 2018.
[48] Xilinx Inc., ‘Vivado Design Suite User Guide, Programming and Debugging - UG908 (v2014.1)’. Xilinx Inc., 2014.
[49] Xilinx Inc., ‘Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics, Product Specification - DS182 (v2.16.1)’. Xilinx Inc., 2018.
[50] Xilinx Inc., ‘Command Line Tools User Guide - UG628 (v 14.5)’. Xilinx Inc., 2013.
[51] R. Jayaraman, ‘Physical Design for FPGAs’, in *Proceedings of the 2001 International Symposium on Physical Design*, New York, NY, USA, 2001, pp. 214–221.
[52] C. Kao, ‘Benefits of Partial Reconfiguration Benefits of Partial Reconfiguration’, *Xcell Journal*, vol. Fourth Quarter, no. 55, pp. 65–67, 2005.
[53] D. Koch *et al.*, ‘Partial reconfiguration on FPGAs in practice — Tools and applications’, in *ARCS 2012*, 2012, pp. 1–12.
[54] E. Eto, ‘Difference-Based Partial Reconfiguration - XAPP290 (v2.0)’. 2007.

[55] T. Wollinger, J. Guajardo, and C. Paar, ‘Security on FPGAs: State-of-the-art Implementations and Attacks’, *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 3, pp. 534–574, Aug. 2004.
[56] A. Telikepalli, ‘Is Your FPGA Design Secure?’, *Xcell Journal*, vol. Fall 2003, no. 47, pp. 32–35, 2003.
[57] W. F. Ehrsam, C. H. W. Meyer, J. L. Smith, and W. L. Tuchman, ‘Message verification and transmission error detection by block chaining’, US4074066A, 14-Feb-1978.
[58] S. Drimer, ‘Authentication of FPGA Bitstreams: Why and How’, in *Reconfigurable Computing: Architectures, Tools and Applications*, P. C. Diniz, E. Marques, K. Bertels, M. M. Fernandes, and J. M. P. Cardoso, Eds. Springer Berlin Heidelberg, 2007, pp. 73–84.
[59] Xilinx Inc., ‘Virtex-4 FPGA Configuration User Guide - UG071 (v1.12)’. 02-Jun-2017.
[60] P. Koopman and T. Chakravarty, ‘Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks’, in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, Washington, DC, USA, 2004, pp. 145–154.
[61] B. Badrignans, F. Devic, L. Torres, G. Sassatelli, and P. Benoit, ‘Embedded Systems Security for FPGA’, in *Security Trends for FPGAS*, B. Badrignans, J. L. Danger, V. Fischer, G. Gogniat, and L. Torres, Eds. Springer Netherlands, 2011, pp. 137–187.
[62] Xilinx Inc., ‘Soft Error Mitigation Controller v4.1 - LogiCORE IP Product Guide’. Xilinx Inc., 2015.
[63] W. E. Cory, D. P. Schultz, and S. P. Young, ‘Error checking parity and syndrome of a block of data with relocated parity bits’, US7426678 B1, 16-Sep-2008.
[64] E. Peterson, ‘Developing Tamper Resistant Designs with Xilinx Virtex-6 and 7 Series FPGAs, Application Note - XAPP1084 (v1.3)’. 2013.
[65] M. Hally, *Electronic Brains: Stories from the Dawn of the Computer Age*. National Academies Press, 2005.
[66] E. W. Pugh, *Building IBM: Shaping an Industry and Its Technology*. MIT Press, 1995.
[67] F. Faggin, ‘The Birth of the Microprocessor’, *BYTE*, vol. 17, no. 3, pp. 145–150, Mar. 1992.
[68] N. Telle, W. Luk, and R. C. C. Cheung, ‘Customising Hardware Designs for Elliptic Curve Cryptography’, in *Computer Systems: Architectures, Modeling, and Simulation*, 2004, pp. 274–283.
[69] G. Stitt, F. Vahid, and S. Nematbakhsh, ‘Energy Savings and Speedups from Partitioning Critical Software Loops to Hardware in Embedded Systems’, *ACM Trans Embed Comput Syst*, vol. 3, no. 1, pp. 218–232, Feb. 2004.
[70] Xilinx Inc., ‘Zynq-7000 All Programmable SoC Overview - Product Specification - DS190 (v1.8)’. Xilinx Inc., 2015.
[71] Altera Corp., ‘Altera’s User-Customizable ARM-Based SoC’. 2015.
[72] G. Estrin, ‘Organization of Computer Systems: The Fixed Plus Variable Structure Computer’, in *Papers Presented at the May 3-5, 1960, Western Joint IRE-AIEE-ACM Computer Conference*, New York, NY, USA, 1960, pp. 33–40.

[73] G. Estrin, ‘Reconfigurable computer origins: the UCLA fixed-plus-variable (F+V) structure computer’, *IEEE Ann. Hist. Comput.*, vol. 24, no. 4, pp. 3–9, Oct. 2002.

[74] ‘Field Programmable Gate Array (FPGA) Market 2018 Global Analysis, Opportunities and Forecast To 2023’, *MarketWatch*, 14-Jun-2018. [Online]. Available: https://www.marketwatch.com/press-release/field-programmable-gate-array-fpga-market-2018-global-analysis-opportunities-and-forecast-to-2023-2018-06-14. [Accessed: 14-Sep-2018].

[75] R. Hartenstein, ‘Are we really ready for the breakthrough? [morphware]’, in *Proceedings International Parallel and Distributed Processing Symposium*, 2003, pp. 7 pp.-.

[76] J.-Y. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, ‘Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip’, in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, 2003, pp. 986–991.

[77] D. Andrews *et al.*, ‘HThreads: a hardware/software co-designed multithreaded RTOS kernel’, in *10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005*, 2005, vol. 2, pp. 331–338.

[78] W. Peck, E. Anderson, J. Agron, J. Stevens, F. Baijot, and D. Andrews, ‘Hthreads: A Computational Model for Reconfigurable Devices’, in *2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–4.

[79] X. Iturbe *et al.*, ‘R3TOS: A novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on FPGAs’, *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1542–1556, Aug. 2013.

[80] H. Kwok-Hay So, ‘BORPH: An Operating System for FPGA-Based Reconfigurable Computers’, University of California, Berkeley, 2007.

[81] E. Lubbers and M. Platzner, ‘ReconOS: An RTOS Supporting Hard-and Software Threads’, in *2007 International Conference on Field Programmable Logic and Applications*, 2007, pp. 441–446.

[82] A. Agne *et al.*, ‘ReconOS: An Operating System Approach for Reconfigurable Computing’, *IEEE Micro*, vol. 34, no. 1, pp. 60–71, Jan. 2014.

[83] D. Gohringer, M. Hubner, E. N. Zeutebouo, and J. Becker, ‘CAP-OS: Operating system for runtime scheduling, task mapping and resource management on reconfigurable multiprocessor architectures’, in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW)*, 2010, pp. 1–8.

[84] A. Ismail and L. Shannon, ‘FUSE: Front-End User Framework for O/S Abstraction of Hardware Accelerators’, in *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2011, pp. 170–177.

[85] Y. Wang *et al.*, ‘SPREAD: A Streaming-Based Partially Reconfigurable Architecture and Programming Model’, *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 21, no. 12, pp. 2179–2192, Dec. 2013.

[86] G. Charitopoulos, I. Koidis, K. Papadimitriou, and D. Pnevmatikatos, ‘Hardware Task Scheduling for Partially Reconfigurable FPGAs’, in *Applied Reconfigurable Computing*, 2015, pp. 487–498.

[87] A. Wold, A. Agne, and J. Torresen, ‘Relocatable Hardware Threads in Run-Time Reconfigurable Systems’, in *Reconfigurable Computing: Architectures, Tools, and Applications*, 2014, pp. 61–72.

[88] K. Jozwik, S. Honda, M. Edahiro, H. Tomiyama, and H. Takada, ‘Rainbow: An Operating System for Software-Hardware Multitasking on Dynamically Partially Reconfigurable FPGAs’, *International Journal of Reconfigurable Computing*, 2013. [Online]. Available: https://www.hindawi.com/journals/ijrc/2013/789134/abs/. [Accessed: 17-Sep-2018].

[89] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, and J. Teich, ‘The Erlangen Slot Machine: a highly flexible FPGA-based reconfigurable platform’, in *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2005. FCCM 2005*, 2005, pp. 319–320.

[90] X. Iturbe, K. Benkrid, T. Arslan, R. Torrego, and I. Martinez, ‘Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs’, in *2011 International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 295–300.

[91] A. W. Wieder and F. Neppl, ‘CMOS technology trends and economics’, *IEEE Micro*, vol. 12, no. 4, pp. 10–19, Aug. 1992.

[92] S. Ferrera and N. P. Carter, ‘Reconfigurable Circuits Using Hybrid Hall Effect Devices’, in *Field Programmable Logic and Application*, 2003, pp. 1–10.

[93] T. S. Nidhin, A. Bhattacharyya, R. P. Behera, T. Jayanthi, and K. Velusamy, ‘Understanding radiation effects in SRAM-based field programmable gate arrays for implementing instrumentation and control systems of nuclear power plants’, *Nucl. Eng. Technol.*, vol. 49, no. 8, pp. 1589–1599, Dec. 2017.

[94] J. Hussein and G. Swift, ‘Mitigating Single-Event Upsets (White Paper - WP395)’. Xilinx Inc., 2015.

[95] T. Buerkle *et al.*, ‘Ionizing Radiation Detector for Environmental Awareness in FPGA-Based Flight Computers’, *IEEE Sens. J.*, vol. 12, no. 6, pp. 2229–2236, Jun. 2012.

[96] M. Nicolaidis, ‘Design for soft error mitigation’, *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 405–418, Sep. 2005.

[97] R. C. Baumann, ‘Radiation-induced soft errors in advanced semiconductor technologies’, *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 305–316, Sep. 2005.

[98] G. Allen, G. Swift, and C. Carmichael, ‘Virtex-4QV Static SEU Characterization Summary’. JPL Publication 08-16 4/08. NASA Jet Propulsion Laboratory, Pasadena, CA, 2008.

[99] P. Adell, G. Allen, G. Swift, and S. McClure, ‘Assessing and mitigating radiation effects in Xilinx SRAM FPGAs’, in *2008 European Conference on Radiation and Its Effects on Components and Systems*, 2008, pp. 418–424.

[100] F. W. Sexton, ‘Destructive single-event effects in semiconductor devices and ICs’, *IEEE Trans. Nucl. Sci.*, vol. 50, no. 3, pp. 603–621, Jun. 2003.

[101] P. Graham, M. Caffrey, J. Zimmerman, P. Sundararajan, E. Johnson, and C. Patterson, ‘Consequences and categories of SRAM FPGA configuration

SEUs', in *In Proceedings of the International Conference on Military and Aerospace Programmable Logic Devices*, 2003, pp. 1–9.
[102] M. Wirthlin, 'High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond', *Proc. IEEE*, vol. 103, no. 3, pp. 379–389, Mar. 2015.
[103] S. Srinivasan *et al.*, 'Toward Increasing FPGA Lifetime', *IEEE Trans. Dependable Secure Comput.*, vol. 5, no. 2, pp. 115–127, Apr. 2008.
[104] L. Kirischian, V. Kirischian, and D. Sharma, 'Mitigation of Thermo-cycling effects in Flip-chip FPGA-based Space-borne Systems by Cyclic On-chip Task Relocation', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 17–24.
[105] R. Katz *et al.*, 'Radiation effects on current field programmable technologies', *IEEE Trans. Nucl. Sci.*, vol. 44, no. 6, pp. 1945–1956, Dec. 1997.
[106] F. Kastensmidt and P. Rech, 'Radiation Effects and Fault Tolerance Techniques for FPGAs and GPUs', in *FPGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design*, F. Kastensmidt and P. Rech, Eds. Cham: Springer International Publishing, 2016, pp. 3–17.
[107] P. Mangalagiri, S. Bae, R. Krishnan, Y. Xie, and V. Narayanan, 'Thermal-aware reliability analysis for platform FPGAs', in *Int'l Conf. Computer Aided Design*, 2008, pp. 722–727.
[108] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, 'Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications', *ACM Comput Surv*, vol. 47, no. 2, pp. 37:1–37:34, Jan. 2015.
[109] A. Camplani, S. Shojaii, H. Shrimali, A. Stabile, and V. Liberali, 'CMOS IC radiation hardening by design', *Facta Univ. - Ser. Electron. Energ.*, vol. 27, no. 2, pp. 251–258, 2014.
[110] R. Roosta, 'A Comparison of Radiation-Hard and Radiation-Tolerant FPGAs for Space Applications'. JPL D-31228. NASA Electronic Parts and Packaging Program, 30-Dec-2004.
[111] D. G. Mavis and D. R. Alexander, 'Employing radiation hardness by design techniques with commercial integrated circuit processes', in *16th DASC. AIAA/IEEE Digital Avionics Systems Conference. Reflections to the Future. Proceedings*, 1997, vol. 1, pp. 2.1-15-2.1-22.
[112] F. Faccio, 'Radiation Effects and Hardening by Design in CMOS Technologies', in *Analog Circuit Design: Robust Design, Sigma Delta Converters, RFID*, H. Casier, M. Steyaert, and A. H. M. van Roermund, Eds. Dordrecht: Springer Netherlands, 2011, pp. 69–87.
[113] J. P. Hayes, I. Polian, and B. Becker, 'An Analysis Framework for Transient-Error Tolerance', in *25th IEEE VLSI Test Symposium (VTS'07)*, 2007, pp. 249–255.
[114] R. Garg, N. Jayakumar, S. P. Khatri, and G. S. Choi, 'Circuit-Level Design Approaches for Radiation-Hard Digital Electronics', *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 17, no. 6, pp. 781–792, Jun. 2009.
[115] R. Ladbury and R. Ladbury, 'Radiation Hardening at the System Level'. IEEE NSREC Short Course, 2007.
[116] J. Von Neumann, 'Probabilistic logics and the synthesis of reliable organisms from unreliable components', in *Automata Studies*, vol. 34, C. E. Shannon and J. McCarthy, Eds. NJ, Princeton: Princeton Univ. Press, 1956, pp. 43–98.

[117] R. E. Lyons and W. Vanderkulk, 'The Use of Triple-Modular Redundancy to Improve Computer Reliability', *IBM J. Res. Dev.*, vol. 6, no. 2, pp. 200–209, Apr. 1962.
[118] S. Habinc, 'Functional Triple Modular Redundancy (FTMR)', Gaisler Research, Dec. 2002.
[119] P. K. Samudrala, J. Ramos, and S. Katkoori, 'Selective Triple Modular Redundancy (STMR) Based Single-Event Upset (SEU) Tolerant Synthesis for FPGAs', *IEEE Trans. Nucl. Sci.*, vol. 51, no. 5, pp. 2957–2969, Oct. 2004.
[120] P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, and J. A. Felix, 'Current and Future Challenges in Radiation Effects on CMOS Electronics', *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 1747–1763, Aug. 2010.
[121] B. Bridgford, C. Carmichael, and C. W. Tseng, 'Single Event Upset Mitigation Selection Guide'. 2008.
[122] A. Lesea and P. Alfke, 'Xilinx FPGAs Overcome the Side Effects of Sub-40 nm Technology'. Xilinx Inc., 2011.
[123] D. S. Lee, G. Swift, and M. Wirthlin, 'An Analysis of High-Current Events Observed on Xilinx 7-Series and Ultrascale Field-Programmable Gate Arrays', in *2016 IEEE Radiation Effects Data Workshop (REDW)*, 2016, pp. 1–5.
[124] T. Bates and C. P. Bridges, 'Single event mitigation for Xilinx 7-series FPGAs', in *2018 IEEE Aerospace Conference*, Big Sky, MT, 2018, pp. 1–12.
[125] A. Stoddard, A. Gruwell, P. Zabriskie, and M. J. Wirthlin, 'A Hybrid Approach to FPGA Configuration Scrubbing', *IEEE Trans. Nucl. Sci.*, vol. 64, no. 1, pp. 497–503, Jan. 2017.
[126] M. Wirthlin, D. Lee, G. Swift, and H. Quinn, 'A Method and Case Study on Identifying Physically Adjacent Multiple-Cell Upsets Using 28-nm, Interleaved and SECDED-Protected Arrays', *IEEE Trans. Nucl. Sci.*, vol. 61, no. 6, pp. 3080–3087, Dec. 2014.
[127] R. Mall, *Real-Time Systems: Theory and Practice*. Pearson Education India, 2009.
[128] G. Swift and G. Allen, 'Virtex-5QV Static SEU Characterization Summary'. Technical Report. NASA Jet Propulsion Laboratory, Pasadena, CA., 2012.
[129] S. Dhingra, D. Milton, and C. E. Stroud, 'BIST for Logic and Memory Resources in Virtex-4 FPGAs', in *Proceedings of the IEEE North Atlantic Test Workshop*, 2006, pp. 19–27.
[130] A. Ebrahim, T. Arslan, and X. Iturbe, 'A fast and scalable FPGA damage diagnostic service for R3TOS using BIST cloning technique', in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–4.
[131] H. Modi and P. Athanas, 'In-system testing of Xilinx 7-Series FPGAs: Part 1-logic', in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, 2015, pp. 477–482.
[132] J. Liu and S. Simmons, 'BIST-diagnosis of interconnect fault locations in FPGA's', in *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436)*, 2003, vol. 1, pp. 207–210 vol.1.

[133] E. Stott and P. Y. K. Cheung, ‘Improving FPGA Reliability with Wear-Levelling’, in *2011 21st International Conference on Field Programmable Logic and Applications*, 2011, pp. 323–328.
[134] S. M. Trimberger, ‘Utilizing multiple test bitstreams to avoid localized defects in partially defective programmable integrated circuits’, US7424655B1, 09-Sep-2008.
[135] W.-J. Huang and E. J. McCluskey, ‘Column-Based Precompiled Configuration Techniques for FPGA’, in *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM’01)*, 2001, pp. 137–146.
[136] J. M. Emmert and D. Bhatia, ‘Incremental Routing in FPGAs’, in *Proceedings Eleventh Annual IEEE International ASIC Conference (Cat. No.98TH8372)*, 1998, pp. 217–221.
[137] S. Dutt, V. Shanmugavel, and S. Trimberger, ‘Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays’, in *1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers (Cat. No.99CH37051)*, 1999, pp. 173–176.
[138] X. She and M. Zwolinski, ‘A novel self-routing reconfigurable fault-tolerant cell array’, in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, 2007, pp. 725–731.
[139] L. Bozzoli and L. Sterpone, ‘Self rerouting of dynamically reconfigurable SRAM-based FPGAs’, in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 77–84.
[140] P. Sedcole, B. Blodget, J. Anderson, P. Lysaghi, and T. Becker, ‘Modular partial reconfigurable in Virtex FPGAs’, in *International Conference on Field Programmable Logic and Applications, 2005.*, 2005, pp. 211–216.
[141] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, ‘Designing an operating system for a heterogeneous reconfigurable SoC’, in *Proceedings International Parallel and Distributed Processing Symposium*, 2003, pp. 7 pp.-.
[142] E. L. Horta and J. W. Lockwood, ‘PARBIT: A Tool to Transform Bitfiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs)’, Washington University, Saint Louis, 2001.
[143] H. Kalte, G. Lee, M. Porrmann, and U. Ruckert, ‘REPLICA: A Bitstream Manipulation Filter for Module Relocation in Partial Reconfigurable Systems’, in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, pp. 151b–151b.
[144] H. Kalte and M. Porrmann, ‘REPLICA2Pro: task relocation by bitstream manipulation in virtex-II/Pro FPGAs’, in *Proceedings of the 3rd conference on Computing frontiers - CF ’06*, Ischia, Italy, 2006, p. 403.
[145] S. Ferrandi. Corbetta, M. Morandi, M. Novati, M. D. Santambrogio, and D. Sciuto, ‘Two Novel Approaches to Online Partial Bitstream Relocation in a Dynamically Reconfigurable System’, in *IEEE Computer Society Annual Symposium on VLSI, 2007. ISVLSI ’07*, 2007, pp. 457–458.
[146] S. Corbetta, M. Morandi, M. Novati, M. D. Santambrogio, D. Sciuto, and P. Spoletini, ‘Internal and External Bitstream Relocation for Partial Dynamic Reconfiguration’, *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 17, no. 11, pp. 1650–1654, Nov. 2009.

[147] A. Sudarsanam, R. Kallam, and A. Dasu, 'PRR-PRR Dynamic Relocation', *Comput. Archit. Lett.*, vol. 8, no. 2, pp. 44–47, Feb. 2009.
[148] M. Hübner, C. Schuck, M. Kiihnle, and J. Becker, 'New 2-dimensional partial dynamic reconfiguration techniques for real-time adaptive microelectronic circuits', in *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06)*, 2006, pp. 6 pp.-.
[149] T. Becker, W. Luk, and P. Y. K. Cheung, 'Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration', in *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007)*, 2007, pp. 35–44.
[150] G. Enemali, A. Adetomi, G. Seetharaman, and T. Arslan, 'A Functionality-Based Runtime Relocation System for Circuits on Heterogeneous FPGAs', *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 65, no. 5, pp. 612–616, May 2018.
[151] A. DeHon, R. Huang, and J. Wawrzynek, 'Hardware-assisted fast routing', in *Proceedings. 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 205–215.
[152] A. A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood, 'OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs', in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 228–235.
[153] D. Koch, C. Beckhoff, and J. Teich, 'ReCoBus-Builder — A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS', in *2008 International Conference on Field Programmable Logic and Applications*, 2008, pp. 119–124.
[154] C. Beckhoff, D. Koch, and J. Torresen, 'Go Ahead: A Partial Reconfiguration Framework', in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012, pp. 37–44.
[155] A. Lalevée, P. H. Horrein, M. Arzel, M. Hübner, and S. Vaton, 'AutoReloc: Automated Design Flow for Bitstream Relocation on Xilinx FPGAs', in *2016 Euromicro Conference on Digital System Design (DSD)*, 2016, pp. 14–21.
[156] X. Iturbe, K. Benkrid, R. Torrego, A. Ebrahim, and T. Arslan, 'Online clock routing in Xilinx FPGAs for high-performance and reliability', in *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 85–91.
[157] M. A. Kadi, P. Rudolph, D. Gohringer, and M. Hubner, 'Dynamic and partial reconfiguration of Zynq 7000 under Linux', in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2013, pp. 1–5.
[158] K. Vipin and S. A. Fahmy, 'ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq', *IEEE Embed. Syst. Lett.*, vol. 6, no. 3, pp. 41–44, Sep. 2014.
[159] F. Duhem, F. Muller, and P. Lorenzini, 'FaRM: Fast Reconfiguration Manager for Reducing Reconfiguration Time Overhead on FPGA', in *Reconfigurable Computing: Architectures, Tools and Applications*, vol. 6578, A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 253–260.

[160] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, 'MetaWire: Using FPGA configuration circuitry to emulate a network-on-chip', *IET Comput. Digit. Tech.*, vol. 4, no. 3, pp. 159–169, May 2010.
[161] J. C. Hoffman and M. S. Pattichis, 'A High-Speed Dynamic Partial Reconfiguration Controller Using Direct Memory Access Through a Multiport Memory Controller and Overclocking with Active Feedback', *Int. J. Reconfigurable Comput.*, pp. 1–10, 2011.
[162] S. Bhandari *et al.*, 'High Speed Dynamic Partial Reconfiguration for Real Time Multimedia Signal Processing', in *2012 15th Euromicro Conference on Digital System Design*, 2012, pp. 319–326.
[163] S. G. Hansen, D. Koch, and J. Torresen, 'High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro', in *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, Los Alamitos, CA, USA, 2011, vol. 0, pp. 174–180.
[164] K. Vipin and S. A. Fahmy, 'A high speed open source controller for FPGA Partial Reconfiguration', in *2012 International Conference on Field-Programmable Technology (FPT)*, 2012, pp. 61–66.
[165] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, 'Multiple-clone configuration of relocatable partial bitstreams in Xilinx Virtex FPGAs', in *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2013, pp. 178–183.
[166] L. A. Cardona and C. Ferrer, 'AC_ICAP: A Flexible High Speed ICAP Controller', *Int. J. Reconfigurable Comput.*, vol. 2015, pp. 1–15, 2015.
[167] A. Ebrahim, K. Benkrid, X. Iturbe, and C. Hong, 'A novel high-performance fault-tolerant ICAP controller', in *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 259–263.
[168] A. Ebrahim, T. Arslan, and X. Iturbe, 'On enhancing the reliability of internal configuration controllers in FPGAs', in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2014, pp. 83–88.
[169] T. S. T. Mak, P. Sedcole, P. Y. K. Cheung, and W. Luk, 'On-FPGA Communication Architectures and Design Factors', in *2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–8.
[170] B. Fu and P. Ampadu, 'Networks-on-Chip (NoC)', in *Error Control for Network-on-Chip Links*, Springer New York, 2012, pp. 33–47.
[171] T. Bjerregaard and S. Mahadevan, 'A Survey of Research and Practices of Network-on-chip', *ACM Comput Surv*, vol. 38, no. 1, Jun. 2006.
[172] V. Adhinarayanan, I. Paul, J. L. Greathouse, W. Huang, A. Pattnaik, and W. c Feng, 'Measuring and modeling on-chip interconnect power on real hardware', in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, 2016, pp. 1–11.
[173] W. J. Dally and B. Towles, 'Route packets, not wires: on-chip interconnection networks', in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684–689.
[174] M. Danashtalab and M. Palesi, 'Basic Concepts on On-Chip Networks', in *Routing Algorithms in Networks-on-Chip*, M. Palesi and M. Daneshtalab, Eds. New York, NY: Springer New York, 2014, pp. 1–18.

[175] É. Cota, A. de M. Amory, and M. S. Lubaszewski, ‘NoC Basics’, in *Reliability, Availability and Serviceability of Networks-on-Chip*, Springer US, 2012, pp. 11–24.
[176] V. Rantala, T. Lehtonen, and J. Plosila, ‘Network on chip routing algorithms’, Turku Centre for Computer Science, 2006.
[177] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.
[178] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, ‘Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs’, in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, M. Glesner, P. Zipf, and M. Renovell, Eds. Springer Berlin Heidelberg, 2002, pp. 795–805.
[179] A. Morgenshtein, I. Cidon, A. Kolodny, and R. Ginosar, ‘Comparative analysis of serial vs parallel links in NoC’, in *2004 International Symposium on System-on-Chip*, 2004, pp. 185–188.
[180] N. Kapre, ‘On Bit-Serial NoCs for FPGAs’, in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 32–39.
[181] F. Alazemi, A. AziziMazreah, B. Bose, and L. Chen, ‘Routerless Network-on-Chip’, in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 492–503.
[182] C. Bobda, M. Majer, D. Koch, A. Ahmadinia, and J. Teich, ‘A Dynamic NoC Approach for Communication in Reconfigurable Devices’, in *Field Programmable Logic and Application*, J. Becker, M. Platzner, and S. Vernalde, Eds. Springer Berlin Heidelberg, 2004, pp. 1032–1036.
[183] M. B. Stensgaard and J. Sparsø, ‘ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology’, in *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, 2008, pp. 55–64.
[184] N. Kapre and J. Gray, ‘Hoplite: Building austere overlay NoCs for FPGAs’, in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.
[185] G. Brebner and A. Donlin, ‘Runtime reconfigurable routing’, in *Parallel and Distributed Processing*, vol. 1388, J. Rolim, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 25–30.
[186] O. Sander, L. Braun, M. Hübner, and J. Becker, ‘Data Reallocation by Exploiting FPGA Configuration Mechanisms’, in *Reconfigurable Computing: Architectures, Tools and Applications*, 2008, pp. 312–317.
[187] M. Welter, ‘Demonstration of Soft Error Mitigation IP and Partial Reconfiguration Capability on Monolithic Devices - XAPP1261 (v1.0)’. Xilinx Inc., 2015.
[188] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, 3rd ed. New York: Springer, 2011.
[189] L. Sha, R. Rajkumar, and J. P. Lehoczky, ‘Priority inheritance protocols: an approach to real-time synchronization’, *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990.
[190] E. Rossi, M. Damschen, L. Bauer, G. Buttazzo, and J. Henkel, ‘Preemption of the Partial Reconfiguration Process to Enable Real-Time Computing With

FPGAs', *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 2, pp. 1–24, Jul. 2018.
[191] M. Damschen, L. Bauer, and J. Henkel, 'CoRQ: Enabling Runtime Reconfiguration Under WCET Guarantees for Real-Time Systems', *IEEE Embed. Syst. Lett.*, vol. 9, no. 3, pp. 77–80, Sep. 2017.
[192] R. Wilhelm *et al.*, 'The worst-case execution-time problem—overview of methods and survey of tools', *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 1–53, Apr. 2008.
[193] J. Sparsø, 'Design of Networks-on-Chip for Real-Time Multi-processor Systems-on-Chip', in *2012 12th International Conference on Application of Concurrency to System Design*, 2012, pp. 1–5.
[194] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. A. E. Ghany, 'Survey on Real-Time Networks-on-Chip', *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1500–1517, May 2017.
[195] K. Papadimitriou, A. Dollas, and S. Hauck, 'Performance of Partial Reconfiguration in FPGA Systems: A Survey and a Cost Model', *ACM Trans Reconfigurable Technol Syst*, vol. 4, no. 4, pp. 36:1–36:24, Dec. 2011.
[196] K. Vipin and S. A. Fahmy, 'FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications', *ACM Comput Surv*, vol. 51, no. 4, pp. 72:1–72:39, Jul. 2018.
[197] A. Adetomi, G. Enemali, and T. Arslan, 'A fault-tolerant ICAP controller with a selective-area soft error mitigation engine', in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 192–199.
[198] M. Cassel and F. Lima, 'Evaluating one-hot encoding finite state machines for SEU reliability in SRAM-based FPGAs', in *12th IEEE International On-Line Testing Symposium (IOLTS'06)*, 2006, pp. 6 pp.-.
[199] Xilinx Inc., '7 Series FPGAs Memory Resources - User Guide UG473 (v1.12)'. Xilinx Inc., 27-Sep-2016.
[200] M. Hikmet, M. M. Kuo, P. S. Roop, and P. Ranjitkar, 'Mixed-Criticality Systems as a Service for Non-critical Tasks', in *2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC)*, 2016, pp. 221–228.
[201] Xilinx Inc., 'UltraScale Architecture Configuration, User Guide - UG570 (v1.9.1)'. Xilinx Inc., 16-Aug-2018.
[202] Xilinx Inc., 'AXI Central Direct Memory Access v4.1 - LogiCORE IP Product Guide (PG034)'. Xilinx Inc., 04-Apr-2018.
[203] A. Ebrahim, 'Dynamic Partial Reconfiguration Management for High Performance and Reliability in FPGAs', University of Edinburgh, Edinburgh, UK.
[204] Xilinx Inc., 'Partial Reconfiguration Controller v1.0 - LogiCORE IP Product Guide (PG193)'. Xilinx Inc., 06-Apr-2016.
[205] Xilinx Inc., 'AXI HWICAP v3.0 - LogiCORE IP Product Guide (PG134)'. 05-Oct-2016.
[206] Xilinx Inc., 'Integrated Logic Analyzer v6.1, LogiCORE IP Product Guide'. Xilinx Inc., 2016.
[207] D. Tamas-Selicean *et al.*, 'Fourier transform spectrometer controller for partitioned architectures', in *2013 IEEE Aerospace Conference*, 2013, pp. 1–11.

[208] R. W. Carlson, K. P. Hand, D. F. Berisford, and D. Keymeulen, 'The Compositional InfraRed Interferometric Spectrometer (CIRIS) for Assessing the Habitability of Europa', *AGU Fall Meet. Abstr.*, vol. 43, p. 2008, Dec. 2013.

[209] X. Iturbe *et al.*, 'Towards a generic and adaptive System-on-Chip controller for space exploration instrumentation', in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2015, pp. 1–8.

[210] X. Iturbe *et al.*, 'Designing a SoC to control the next-generation space exploration flight science instruments', in *2015 28th IEEE International System-on-Chip Conference (SOCC)*, 2015, pp. 13–18.

[211] A. Adetomi, G. Enemali, and T. Arslan, 'Relocating Encrypted Partial Bitstreams by Advance Task Address Loading', in *25th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2017)*, 2017, pp. 188–191.

[212] A. Adetomi, G. Enemali, and T. Arslan, 'Towards an efficient intellectual property protection in dynamically reconfigurable FPGAs', in *2017 Seventh International Conference on Emerging Security Technologies (EST)*, 2017, pp. 150–156.

[213] S.-S. Wang and W.-S. Ni, 'An efficient FPGA implementation of advanced encryption standard algorithm', in *Proceedings of the 2004 International Symposium on Circuits and Systems, 2004. ISCAS '04*, 2004, vol. 2, pp. II-597-600 Vol.2.

[214] M. R. M. Rizk and M. Morsy, 'Optimized area and optimized speed hardware implementations of AES on FPGA', in *Design and Test Workshop, 2007. IDT 2007. 2nd International*, 2007, pp. 207–217.

[215] V. Lomné, A. Dehaboui, P. Maurine, L. Torres, and M. Robert, 'Side Channel Attacks', in *Security Trends for FPGAS*, B. Badrignans, J. L. Danger, V. Fischer, G. Gogniat, and L. Torres, Eds. Springer Netherlands, 2011, pp. 47–72.

[216] A. Moradi, M. Kasper, and C. Paar, 'On the Portability of Side-Channel Attacks – An Analysis of the Xilinx Virtex 4, Virtex 5, and Spartan 6 Bitstream Encryption Mechanism', Cryptology ePrint Archive, 391, 2011.

[217] N. S. S. Srinivas and M. Akramuddin, 'FPGA based hardware implementation of AES Rijndael algorithm for Encryption and Decryption', in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, pp. 1769–1776.

[218] K. Wilkinson, 'Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream'. Xilinx Inc., 2017.

[219] W. Vanderbauwhede, S. R. Chalamalasetti, S. Purohit, and M. Margala, 'A few lines of code, thousands of cores: High-level FPGA programming using vector processor networks', in *2011 International Conference on High Performance Computing Simulation*, 2011, pp. 461–467.

[220] T. Isshiki and W. W. Dai, 'High-Level Bit-Serial Datapath Synthesis for Multi-FPGA Systems', in *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995, pp. 167–173.

[221] A. Adetomi, G. Enemali, and T. Arslan, 'Clock Buffers, Nets, and Trees for On-Chip Communication: A Novel Network Access Technique in FPGAs', in

2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 219–222.

[222] A. Adetomi, G. Enemali, and T. Arslan, 'Relocation-Aware Communication Network for Circuits on Xilinx FPGAs', in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[223] A. Adetomi, G. Enemali, and T. Arslan, 'Characterization of Clock Buffers for On-Chip Inter-Circuit Communication in Xilinx FPGAs', in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[224] A. Adetomi, G. Enemali, G. Seetharaman, and T. Arslan, 'Fault-Tolerant Mechanisms for Relocation-Aware Dynamic On-Chip Communication on FPGAs', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Edinburgh, UK, 2018, p. In Press.

[225] J. Lamoureux and S. J. E. Wilton, 'FPGA Clock Network Architecture: Flexibility vs. Area and Power', in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, 2006, pp. 101–108.

[226] S. Verma and A. S. Dabare, 'Understanding clock domain crossing issues', *EE Times*, 2007.

[227] Xilinx Inc., 'Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide - User Guide UG953 (v2016.2)'. Xilinx Inc., 2016.

[228] W. Simpson, 'PPP in HDLC-like Framing'. [Online]. Available: https://tools.ietf.org/html/rfc1662. [Accessed: 21-Jul-2016].

[229] S. Cheshire and M. Baker, 'Consistent overhead byte stuffing', *IEEEACM Trans. Netw.*, vol. 7, no. 2, pp. 159–172, Apr. 1999.

[230] P. Lin, 'One wire serial communication protocol method and circuit', US7111097B2, 19-Sep-2006.

[231] Xilinx Inc., 'Spartan-7 FPGAs Data Sheet: DC and AC Switching Characteristics, Product Specification - DS189 (v1.7)'. Xilinx Inc., 2018.

[232] Xilinx Inc., 'Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics, Product Specification - DS181 (v1.25)'. Xilinx Inc., 2018.

[233] Xilinx Inc., 'Virtex-7 T and XT FPGAs Data Sheet: DC and AC Switching Characteristics, Product Specification - DS183 (v1.27)'. Xilinx Inc., 2017.

[234] A. Adetomi, G. Enemali, X. Iturbe, D. Keymeulen, and T. Arslan, 'R3TOS-Based Integrated Modular Space Avionics for On-Board Real-Time Data Processing', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Edinburgh, UK, 2018, pp. 1–8.

[235] V. Saptari, *Fourier Transform Spectroscopy Instrumentation Engineering*. SPIE Press, 2004.

[236] M. Koester, W. Luk, J. Hagemeyer, and M. Porrmann, 'Design optimizations to improve placeability of partial reconfiguration modules', in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, 2009, pp. 976–981.

[237] G. Enemali, A. Adetomi, and T. Arslan, 'Expanding the un-usable area strategy for improved utilization of reconfigurable FPGAs', in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 139–144.

[238] G. Enemali, A. Adetomi, and T. Arslan, ‘FAReP: Fragmentation-Aware Replacement Policy for Task Reuse on Reconfigurable FPGAs’, in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 202–206.

[239] G. Enemali, A. Adetomi, and T. Arslan, ‘A placement management circuit for efficient realtime hardware reuse on FPGAs targeting reliable autonomous systems’, in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

Appendices

Appendix A ICAP Access Command Templates

For the various operations of the ICAP controller, configuration command packets are kept inside the IBUF as templates and are written to the ICAP along with the configuration frame data. The commands are based on the registers in Table 2.3. Details on how to compose read and write commands packets have been presented in Section 2.1.2. The benefit of using templates is that the commands are generic and thus reusable, minimizing external storage requirement as incoming bitstreams do not have to include the commands already in these templates.

A.1 Operation Starting Sequence (OSS)

Configuration commands and data are sent to the internal configuration logic only after it has been synchronized by sending a special *Synchronization Word* (0xAA995566) through the ICAP interface. The purpose of this is to enable the alignment of the configuration data with the internal configuration logic. The OSS sequence of Figure A.1 is used to achieve this. After the synchronization, the CRC register is reset by writing 0x00000007 to the CMD register 0x30008001 to restart CRC check coverage. Figure A.2 is a waveform showing the OSS in action. The waveform shows the transition of the ICAP's output from 0xFFFFFFD9 to 0xFFFFFFDB indicating that the configuration interface has been synchronized.

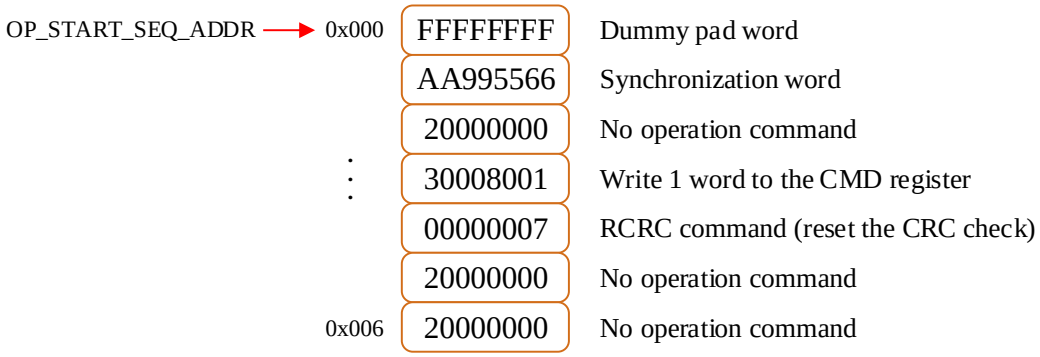


Figure A.1: Operation starting sequence commands

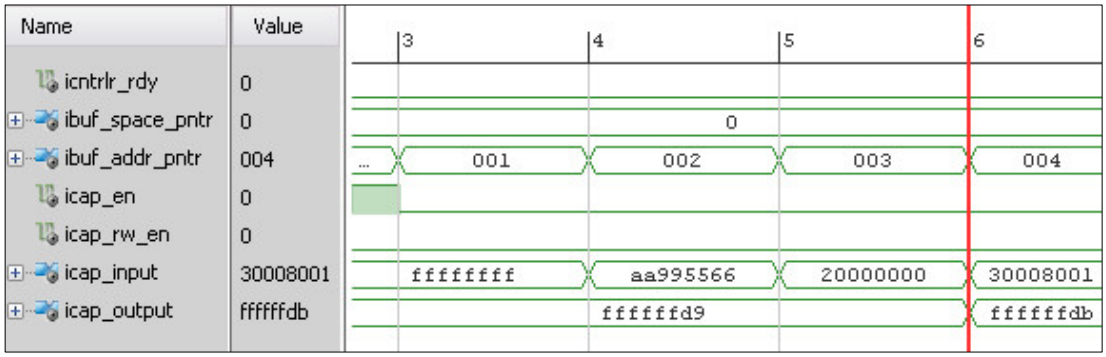


Figure A.2: Waveform showing the synchronization of the configuration interface

A.2 Configuration Frame Readback Template (CFRT)

To perform a readback operation, the IFSM fetches and uses the CFRT template (see Figure A.3) at the address RBK_TEMPLT_ADDR (0x007). In order to read from the CMEM, the RCFG command must be used and a frame address specified. The user is expected to set the FAR value at template address 0x00E. The user content in the LUT-based RAMs could have changed. If a readback operation is performed without masking LUTs, the read back frame data would reflect the user content rather than the original CMEM values. If this is not a desired behavior, the LUTs can be masked by using the GLUTMASK_B bit in the CTL0 register. In fact, this is the default setting. However, the user can control the masking by writing to the template address 0x00A.

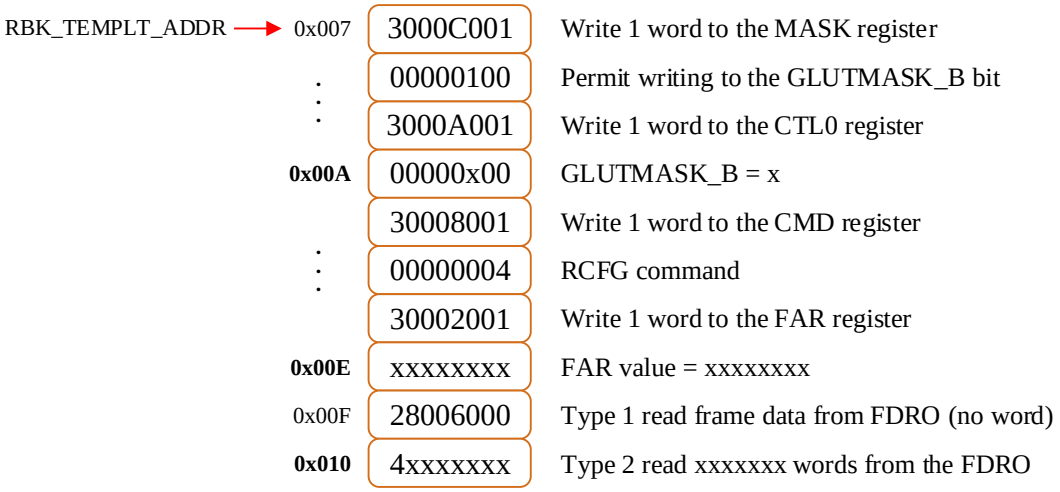


Figure A.3: Configuration frame readback template

A.3 Configuration Frame Write Template (CFWT)

CMEM writing is done by writing a number of contiguous frames to the FDRI register preceded by issuing the WCFG command (0x30008001 followed by 0x00000001). Every write to the CMEM requires a valid device ID, which must be loaded into the IDCODE register. This ID is inserted in the bitstream generated by the design tool. The user should specify this ID at address 0x019 in the IBUF (see the CFWT template in Figure A.4). The user can also set the GLUTMASK_B bit at address 0x015. In the basic form of configuration, every FDRI register loading must be finished with the loading of a pad frame (101 dummy words) to flush the internal pipeline of the configuration logic.

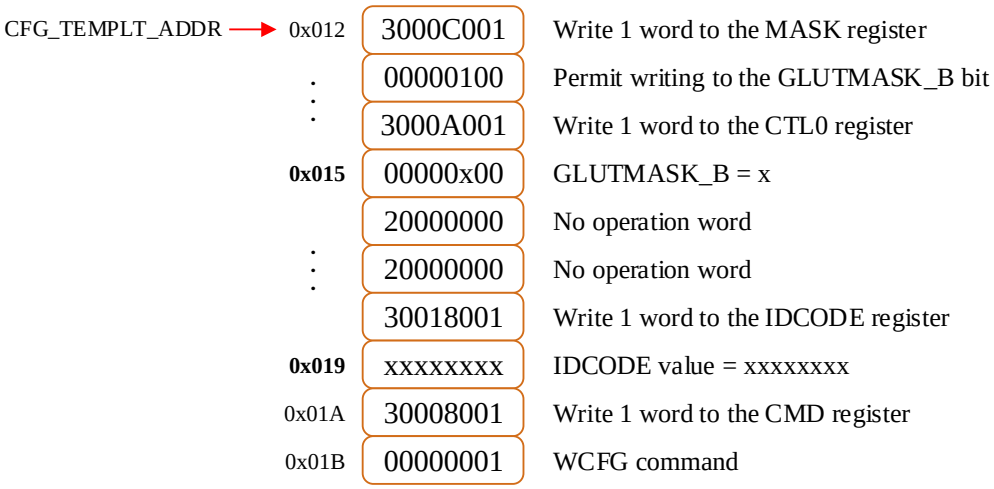


Figure A.4: Configuration frame write template

The CFWT template does not include commands for loading the FAR and FDRI registers because these commands are expected to be included in the bitstream that is being configured, especially since the user may be interested in performing PBR, in which case the original frame address is not of great significance. As such, the task bitstream supplied for (re)configuration has to be in the format presented in Figure A.5 and buffered in the IBUF starting at address 0x073. The preamble and postamble of the bitstream are generally catered for by the OSS and the OES respectively. Nevertheless, additional setup commands not included in the OSS can be added before the 0x30002001command in Figure A.5. Likewise, additional commands can be loaded

after uploading the frame data. It should be noted that a typical bitstream has a multiplicity of frame addresses and the IFSM has been implemented to handle this.

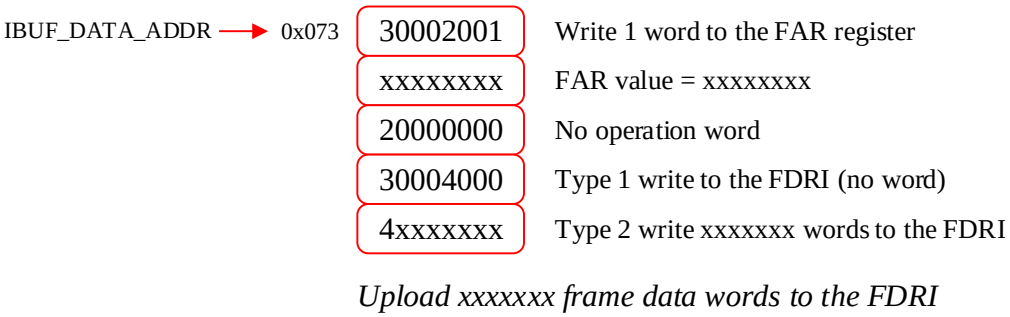


Figure A.5: Configuration command and frame data format for task bitstreams

A.4 Multiple Frame Write Template (MFWT)

The MFWT template provides the commands and parameter placeholders for using the multiple frame write feature of the FPGA. In the basic form of configuration based on the CFWT template, a pad frame is incurred for every write to the CMEM. However, with the MFW functionality, the pad frame is not needed, though the configuration commands and steps are more involving. However, there is an overall benefit of lower configuration time overhead. Meanwhile, the MFW is expected to be done frame-by-frame. This means that for a large bitstream, the traditional configuration method would be better. It should also be noted that though the MFW is intended for writing the same frame data to multiple frame locations especially for bitstream compression, it as well allows a single frame location to be written. Figure A.6 shows the MFW template. The first part of it is the CFWT presented in Figure A.4.

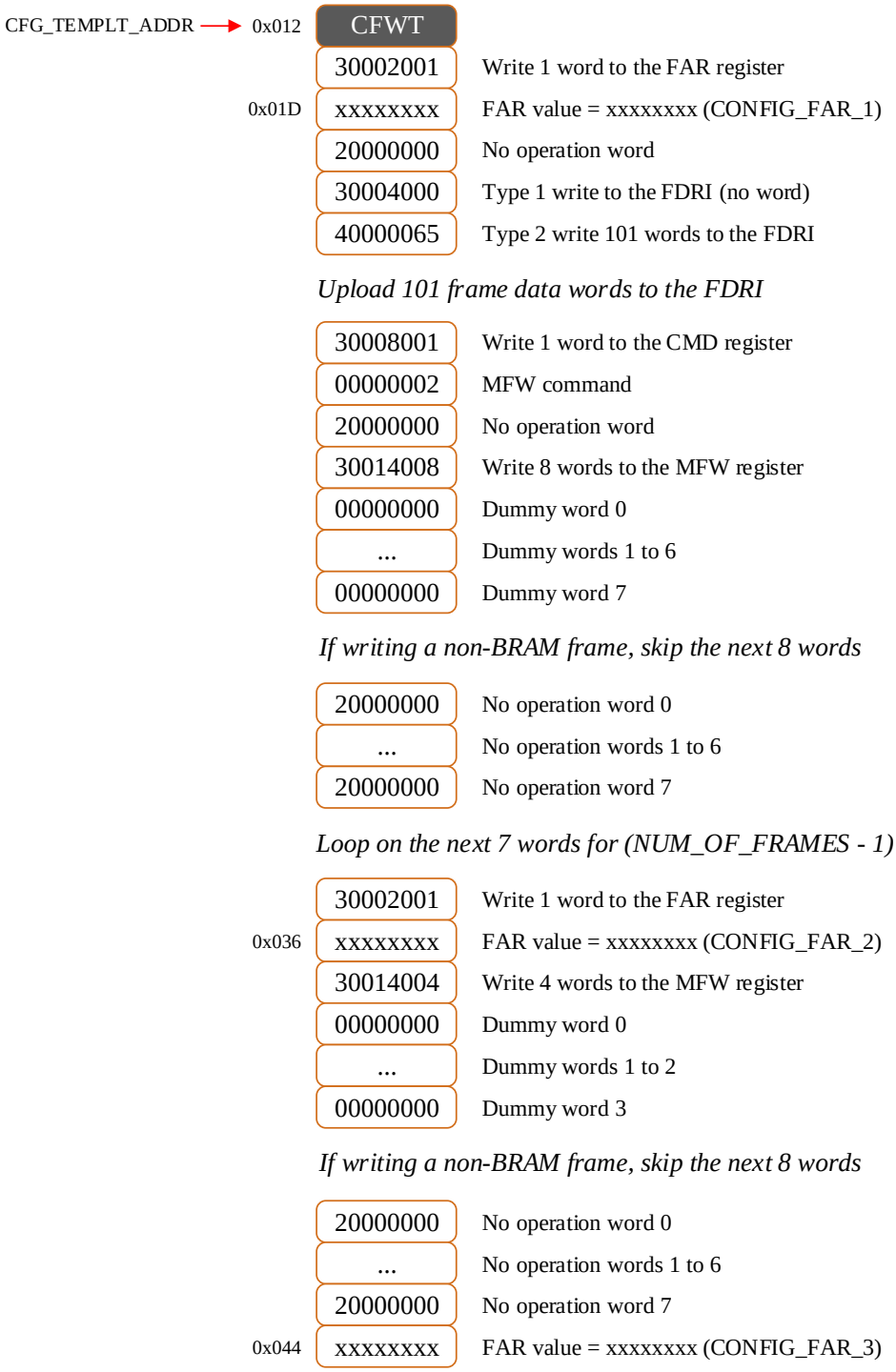


Figure A.6: Multiple frame write template

A.5 Configuration Frame Blanking Template (CFBT)

The frame blanking uses the MFW feature. As such, it relies heavily on the MFW template as shown in Figure A.7.In the use of the CFBT, 101 blank words are not stored in the IBUF. Only a single blank word (0x00000000) is kept at address BLK_DATA_ADDR address of 0x059 and written 101 times when needed. The template requires two parameters, the starting frame address (START_FAR) and the number of frames to write at addresses 0x04A and 0x04D. In addition, since the CFBT is basically a frame write template, the correct IDCODE value should be specified at address 0x046.

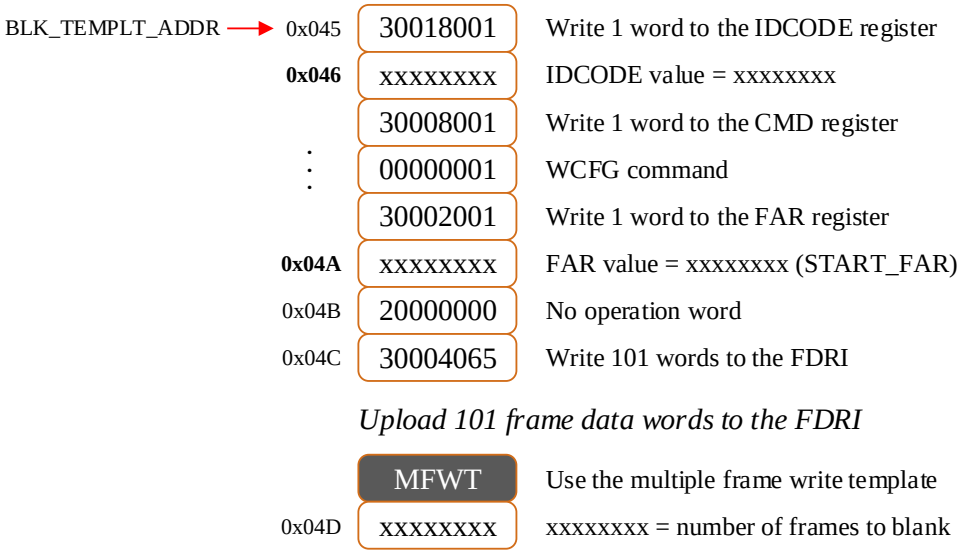


Figure A.7: Configuration frame blanking template

A.6 Operation Ending Sequence (OES)

In this sequence (see Figure A.8), the GLUTMASK_B bit is restored to the default value, which allows the masking of changeable memory cell readback values. This is done regardless of GLUTMASK_B's user setting in the OSS sequence. The next command issued is for resetting the CRC register. At this point in the bitstream loading, the configuration logic expects a precomputed CRC value that it can compare with the one calculated while loading the bitstream data. However, because some of the operations involve a deliberate modification of the bitstream (e.g., PBR), essentially

rendering the precomputed CRC void, resetting the CRC register is necessary. Meanwhile, to restore the bitstream CRC protection, a CRC-32 circuit can easily be used to recompute the CRC on-chip but this has not been implemented in this work.

The last command in the OES sequence is the desynchronization of the configuration interface by writing 0x0000000D to the CMD register. Each time the configuration interface is desynchronized two NOOP words must be written to flush the internal packet buffer.

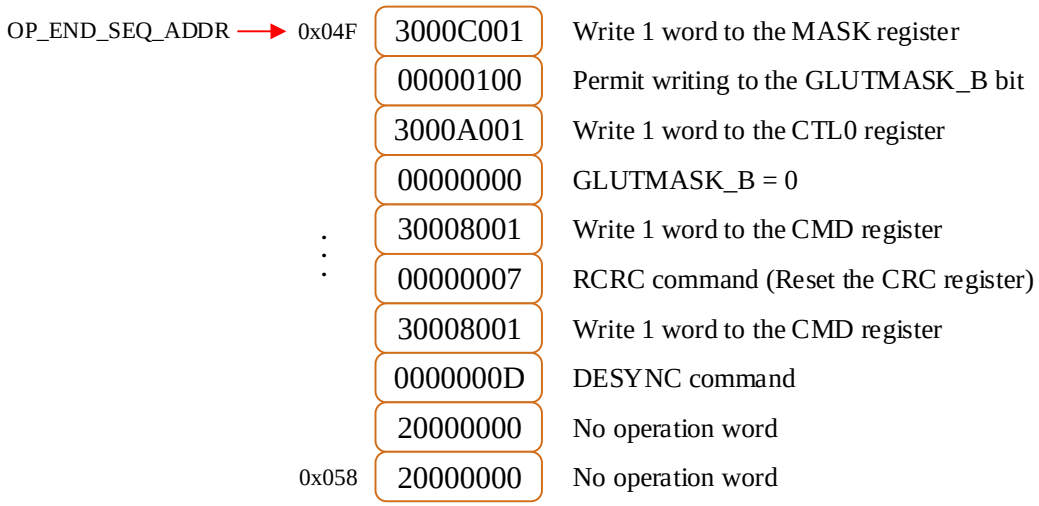


Figure A.8: Operation ending sequence

Appendix B ICAP Access Operation Waveforms

B.1 Readback (RBK) Operation

Figure B.1 is a waveform showing a successful readback operation. The first two *icap_en* pulses indicate temporary disabling of data loading while the IFSM determines the sequences and templates to load, the first pulse being for OSS and the second for CFRT. The last two pulses are for switching the ICAP from write to read and then from read to write, the last writing being for the loading the OES. Note should also be taken of how the pad frame (101 instances of 0x00000000) is discarded by fixing *ibuf_addr_pntr* at 0x073 for 101 clock cycles. In addition, it should be noticed that the configuration interface is successfully desynchronized (*icap_output* becomes 0xFFFFFFD9) after the operation.

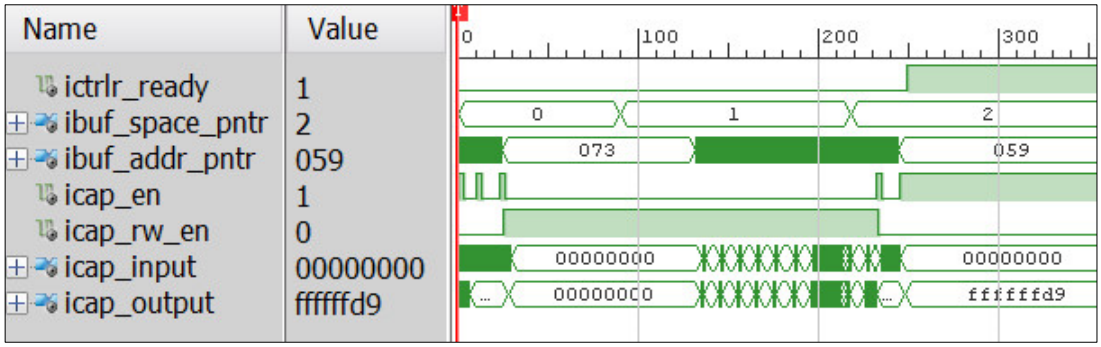


Figure B.1: Waveform showing a readback operation

B.2 Configuration (CFG) Operation

Figure B.2 shows the handover from the CFWT to the user-specified uploaded data during a CFG operation. It should be noted that the ICAP is disabled to carry out this process. Figure B.3 shows the detection of a FAR loading command (0x30002001) and the subsequent modification of the frame address during CFG operation. It should be noted that the loading is paused by only two clock cycles during which a new FAR value is computed. An internal signal (*en_data_ifsm_to_icap*) is used to force the writing of the newly calculated FAR value.

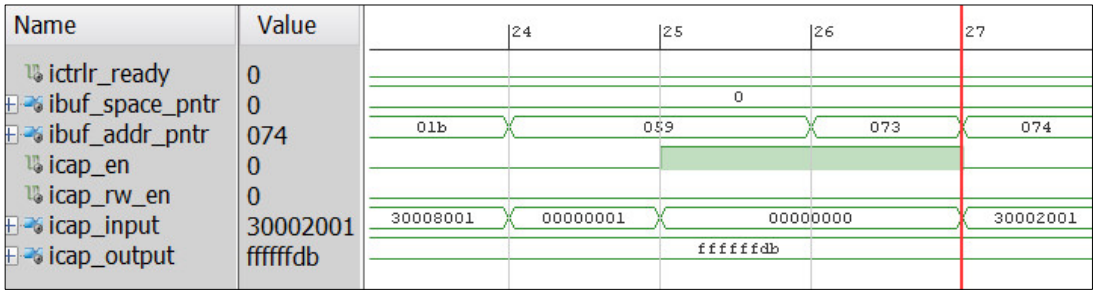


Figure B.2: Waveform showing the handover from the CFWT template to the user data

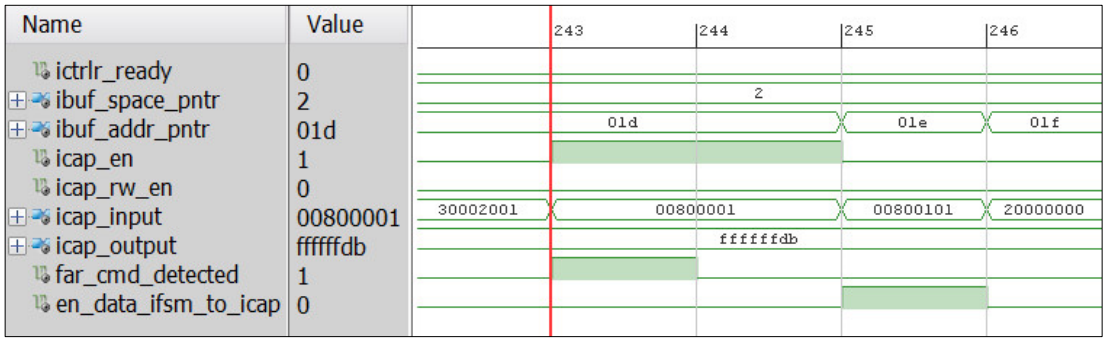


Figure B.3: Waveform showing frame address detection and modification

B.3 Read-Modify-Write (RMW) Operation

Figure B.4 is a waveform showing the read-modify-write of a BRAM content frame. Notice the automatic switching from reading to writing (*icap_rw_en* goes from 1 to 0).

B.4 Blanking (BLK) Operation

Figure B.5 gives an example of the BLK operation on 3 contiguous frames with frame addresses 0x00800101 to 0x00800103. Note the two pulses on the *en_data_ifsm_to_icap* internal signal, which are used to write internally-generated FAR values to the ICAP. Note that the incremented FAR values are determined ahead of when they are needed. As a result, data loading does not stall waiting for the FAR calculation. Note also that the *ibuf_addr_pntr* does not change during the writing of the blank word (0x00000000). This is because the blank word is kept in the IBUF at address 0x059 and written iteratively for 101 times, thereby saving 100 memory spaces in the IBUF.

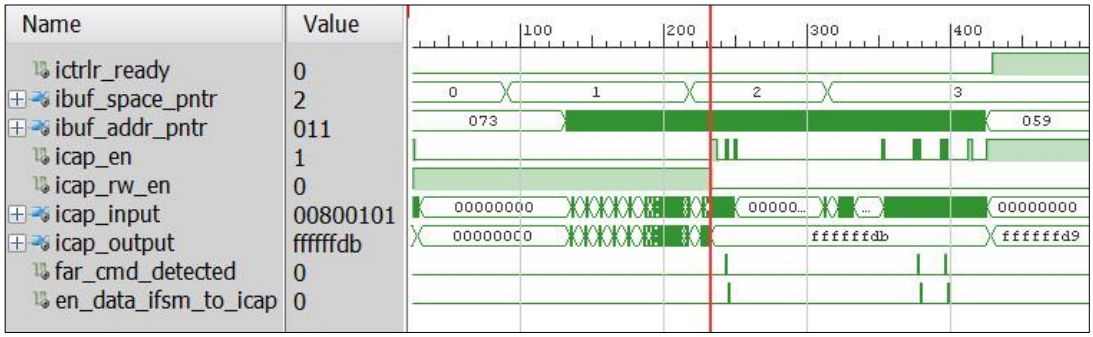


Figure B.4: Waveform showing the RMW operation. Notice the switch from readback to writing at the cursor

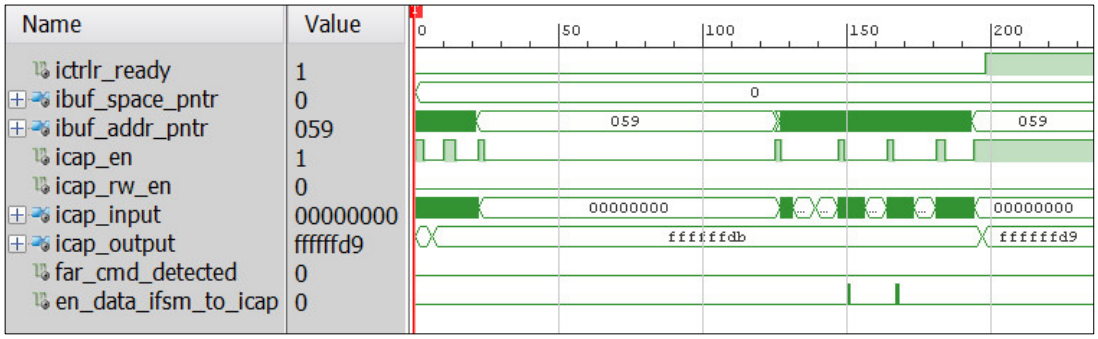


Figure B.5: Waveform showing a blanking operation

B.5 Register Read (RGR) Operation

An example of the RGR operation being used to read the device IDCODE is shown in Figure B.6. In this example, the register address is 0b01100 (see Table 2.3) and the RGR command is therefore 0x28018001. The IDCODE of the used device is correctly read back as 0x0362D093.

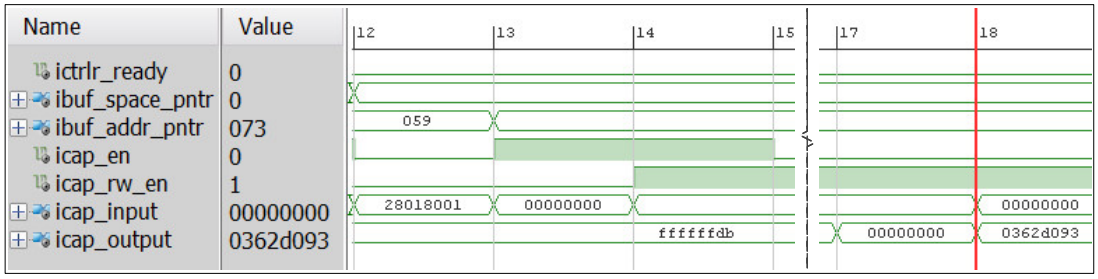


Figure B.6: RGR operation being used to read the device IDCODE

B.6 Abort (ABT) Operation

Figure B.7 is a captured waveform that shows the successful abort of a readback operation and the return of the FSM and the ICAP interface to a stable state, ready for another operation. A corresponding CFG configuration abort is shown in Figure B.8. When an abort is in progress, the bit 4 of the unswapped ICAP's output should read '0' [47]. After *icap_rw_en* is toggled (changed to 0 during a readback operation) in Figure B.7, *icap_output*[7:0] becomes 0xD1. If this is swapped, it gives 0x8B. The 4th bit in 0x8B is a '0', confirming the ABT operation is correctly performed. Notice that the device is eventually desynchronized by the abort process. To resume configuration or readback after an ABT operation, the configuration interface is resynchronized by a new operation which loads the OSS sequence.

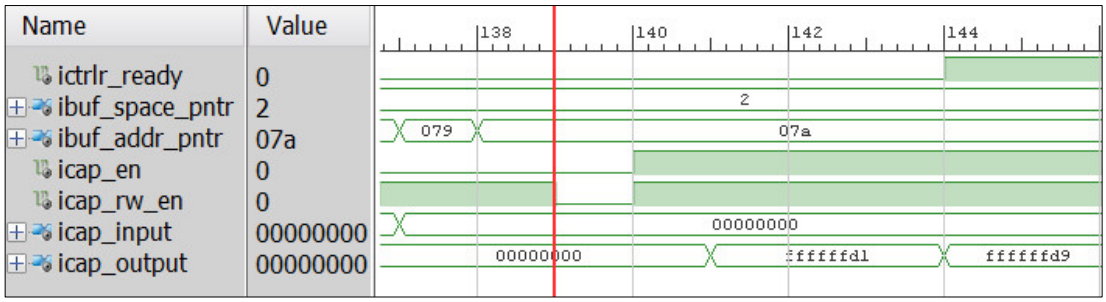


Figure B.7: Waveform showing the successful abort of an RBK operation

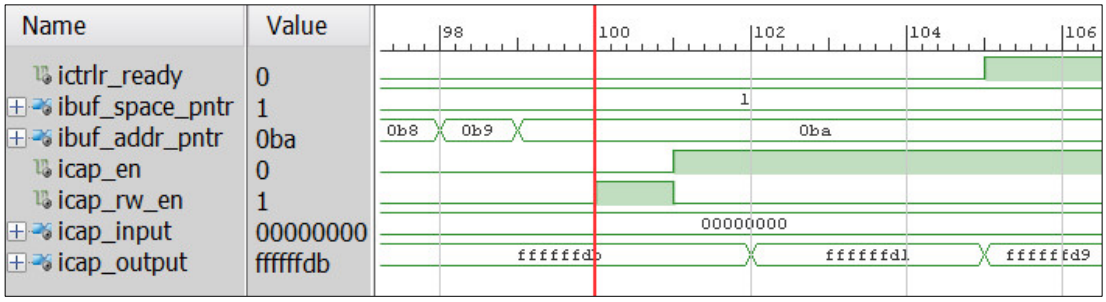


Figure B.8: Waveform showing the successful abort of a CFG operation

B.7 SEM Operation Validation

To validate the SEM operation, a frame (at address 0x00000400) is read back and the 10th bit of the 35th word is flipped and the frame written back by using the CWR (See

Section 4.3.3 where a system-level approach to fault injection analysis is presented). Access to the IBUF to perform this is gained via a VIO connected to the IFSM's control interface and this is the same for all the waveform snapshots in this work. Figure B.9 presents a waveform of the SEM operation in action. A single-bit error is correctly detected at the 10th bit of word 35 as indicated by the internal signals *syn_bit_addr* and *syn_word*. The ecc_*error_far* port also correctly points to the offending FAR address.

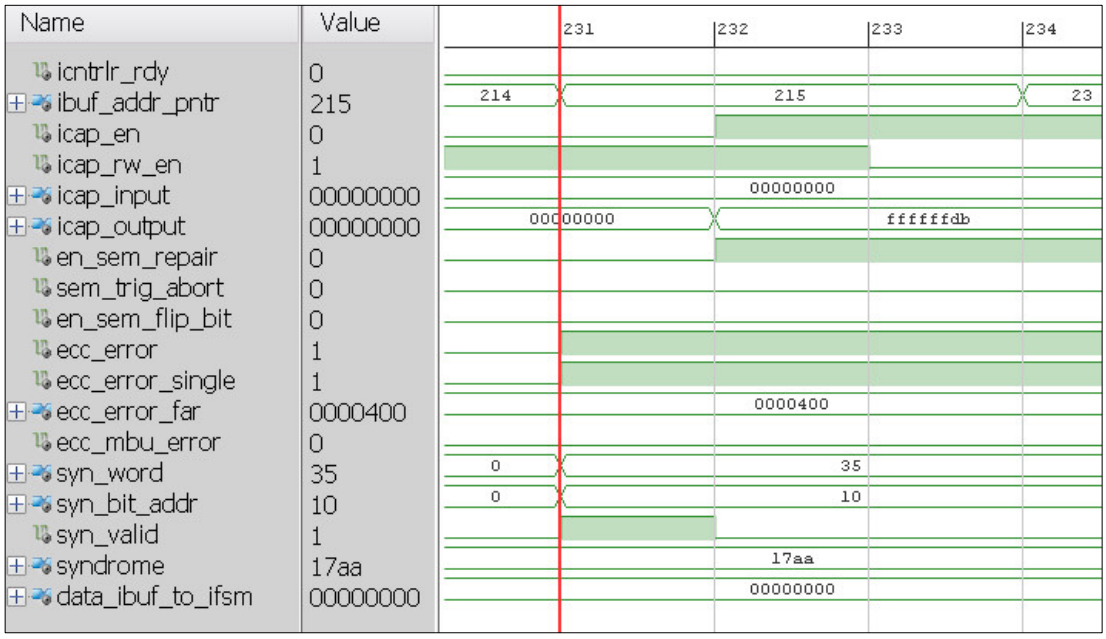


Figure B.9: Waveform showing the detection of a single-bit error by the SEM operation

Figure B.10 shows the correction of the flipped bit while the frame is being written back to the CMEM. As shown, the data from the IBUF (*data_ibuf_to_ifsm*) is 0x00000400 (35th word with a flipped bit in the 10th position). The *icap_input*, however, has been corrected to 0x00000000. Notice the *en_sem_flip_bit* internal signal pulse that controls the flipping combinatorially without stalling the frame data loading. A reissue of the SEM operation does not detect any error.

The same process carried out for the single-bit error is done for a double-bit error. In this case, the error cannot be corrected and is simply reported on the *ecc_mbu_error* port as shown in Figure B.11.

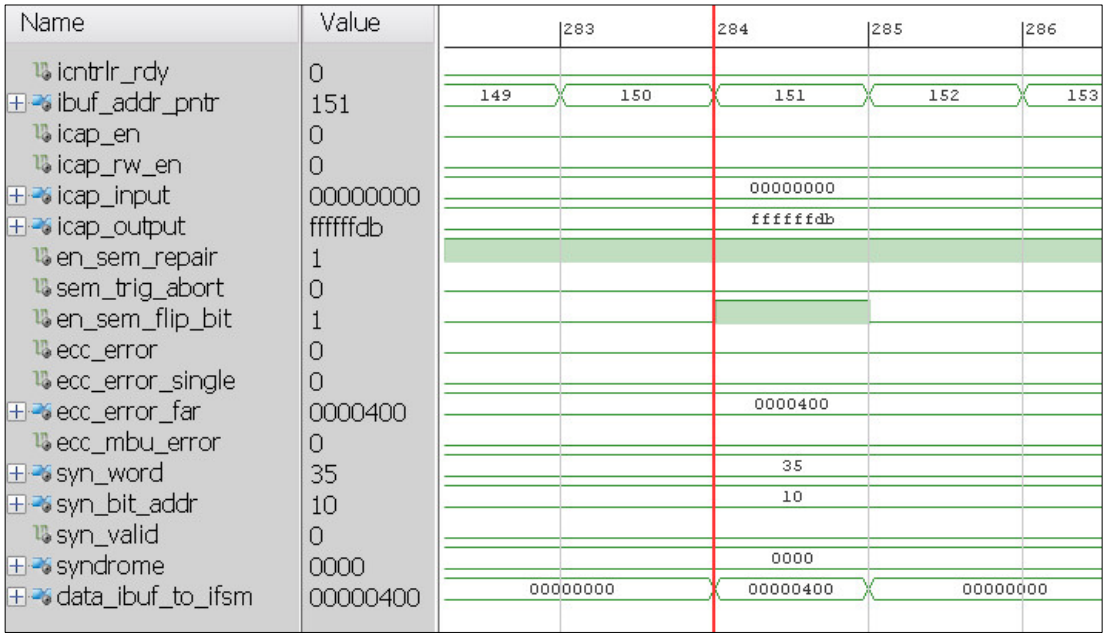


Figure B.10: Waveform showing the correction of an SEU by the SEM operation

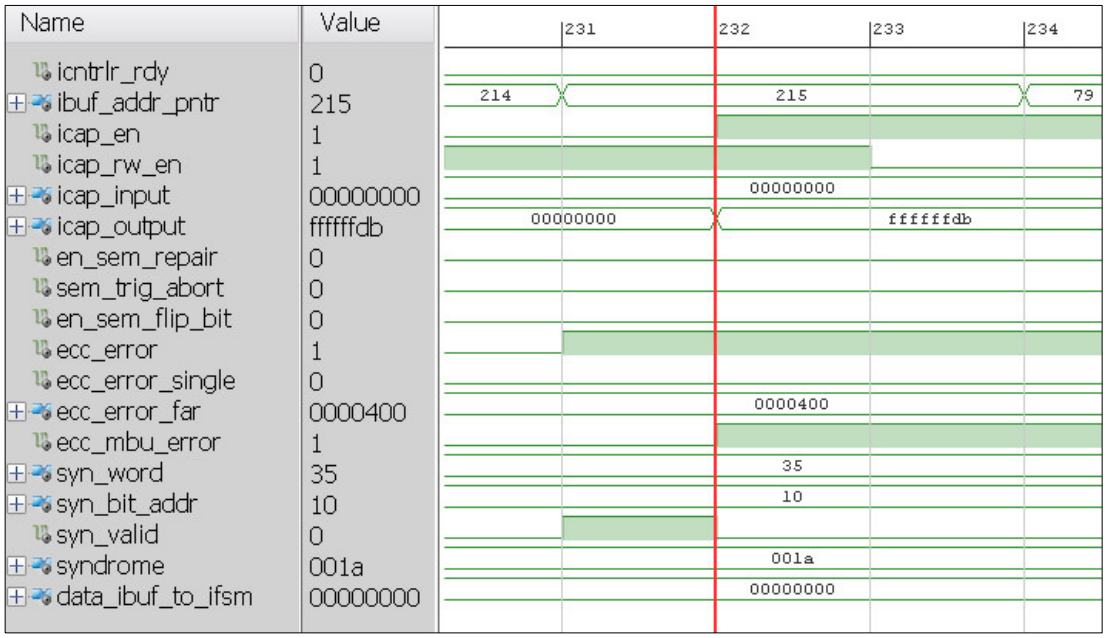


Figure B.11: Waveform showing a multi-bit error detection by the SEM operation

B.8 Configuration Error Monitoring and Recovery

Figure B.12 shows the detection of a configuration error triggered by deliberately loading the CRC register with an incorrect precomputed value using the CWR

operation. Notice the transition of the ICAP output's least significant byte from 0xDB to 0xDA, and finally to 0xD8, signifying that the configuration interface has been desynchronized after an error detection (see Table 4.7). An internal *error_detected* signal is used to trigger the IFSM into action. In Figure B.13, the configuration interface is resynchronized by the IFSM (0xD8 changes to 0xDA) but the error still persists. To prevent the resync process from clearing a CRC error, the RCRC command in the OSS (see Figure A.1 in Appendix A) is avoided by using only the first three words of the OSS. Notice also that the STAT register read command (0x2800E001) to determine the source of error.

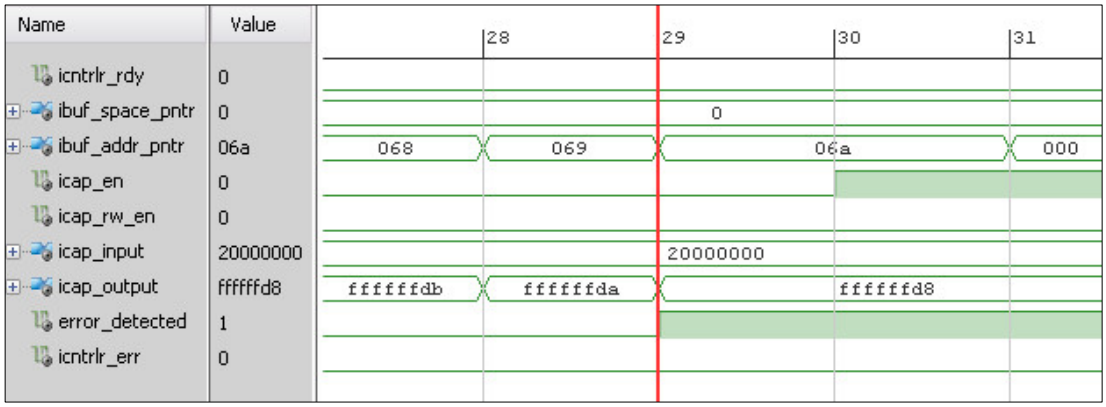


Figure B.12: Waveform showing configuration error detection

Figure B.13: Waveform showing the resynchronization of the configuration interface after configuration error detection

In Figure B.14, the STAT register value is read back and the *icntrlr_err* signal is asserted to alert the user. Notice that the *ibuf_addr_pntr* reflects the fact that the value

read has been written to the IBUF_DATA_ADDR at 0x073. The user can read this location to determine the source of error and act accordingly. The readback STAT register value of 0x46107DFD has bit 0 as a '1' correctly indicating that a CRC error has occurred.

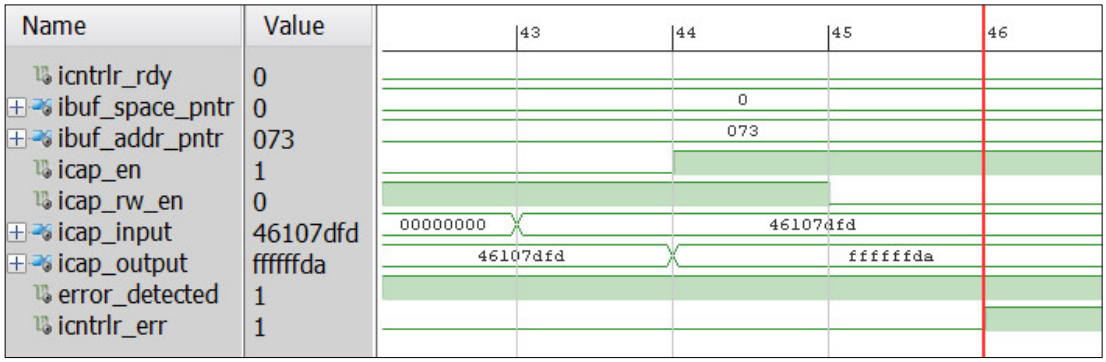


Figure B.14: Waveform showing the STAT register value and the eventual assertion of the icntrlr_err signal after a configuration error detection

Appendix C Splixbit ICAP Controller Waveforms

Figure C.1 shows that Splixbit hardware's IFSM parsing the global preamble of an ATAL-formatted bitstream. Once the number of body parts has been decoded as 2, the IFSM requires the setting of the first FAR value by asserting the *set_new_far* port. Once the new FAR value is available (*new_far_avlbl* pulsed), the frame address (0x00420000) is registered and written in plain format ahead of the encrypted frame data (see Figure C.2). The number of encrypted bytes to load after the plain FAR writing is retrieved from the DWC as 26,976 (see Figure C.3).

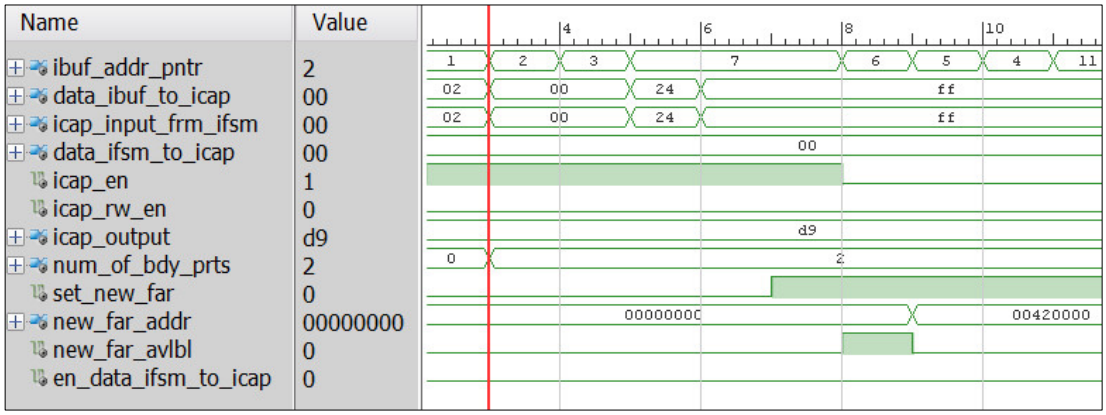


Figure C.1: Waveform showing how the Splixbit's IFSM parses the global preamble and retrieves the plain frame address

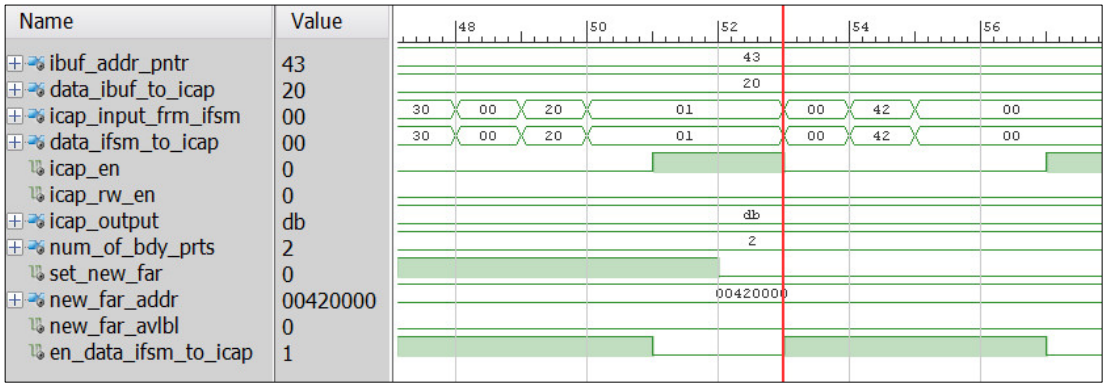


Figure C.2: Waveform showing the online loading of a plain frame address in advance of encrypted frame data

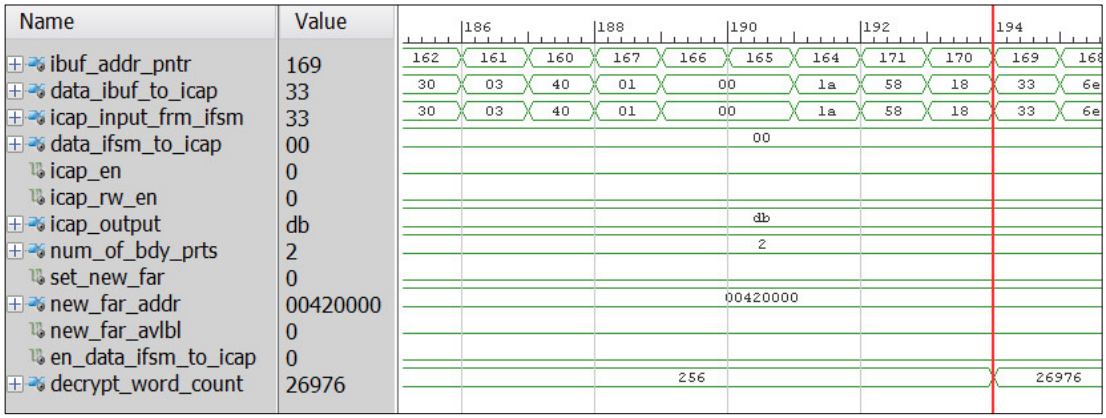


Figure C.3: Waveform showing the decoding of the DWC by the Splixbit's IFSM

Appendix D CONS Waveforms

Figure D.1 and Figure D.2 are respective waveforms showing the non-addressable and address-inclusive modes of the CONS encoding and decoding. Because the *use_addr* signal is driven low in Figure D.1, the *addr_rcvd* port on the decoder remains zero all through the decoding as against in Figure D.2, where it changes to reflect the received plain unencoded address before the decoding begins. The 4-bit address 0b1010 accompanies the packet 0x10900016 and is received before the serial encoded packet is received.

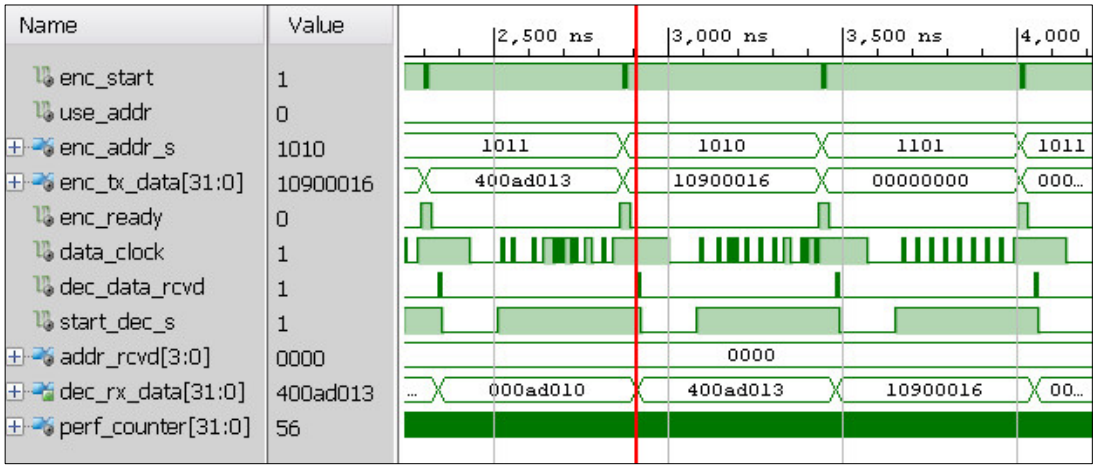


Figure D.1: Waveform showing the non-addressable encoding and decoding of packets

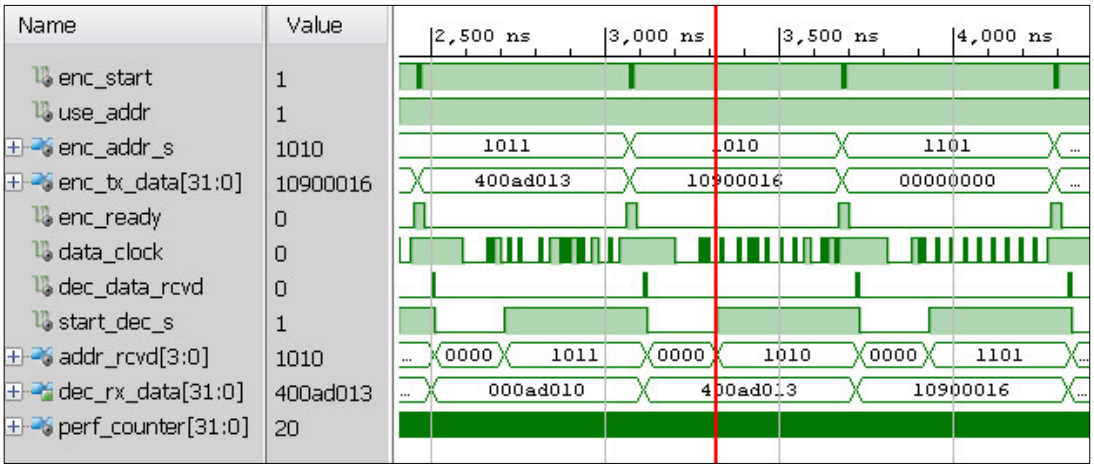


Figure D.2: Waveform showing the addressable encoding and decoding of packets